

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Вебзастосунок для прогнозування ринкових трендів на основі ансамблів
нейромереж»**

Виконав студент групи ІПЗ-22-2
ГАНЧЕВСЬКИЙ Олександр Олександрович

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГАНЧЕВСЬКИЙ Олександр Олександрович**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Вебзастосунок для прогнозування ринкових трендів на основі ансамблів нейромереж»,**

затверджена Наказом закладу вищої освіти від **«14» лютого 2026р., №61/с**

Вихідні дані для роботи: **розробка веб-додатку для прогнозу ринкових трендів.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Вакалюк Т. А.	14.02.2026 р.	14.02.2026 р.
2	Вакалюк Т. А.	14.02.2026 р.	14.02.2026 р.
3	Вакалюк Т. А.	14.02.2026 р.	14.02.2026 р.

РЕФЕРАТ

Випускна робота бакалавра складається з реалізації програмного продукту «Вебзастосунок для прогнозування ринкових трендів на основі ансамблів нейромереж» та пояснювальної записки. Пояснювальна записка до випускної роботи містить 83 сторінок, включає 13 ілюстрацій, 4 таблиці, 1 додатка та 31 використане літературне джерело.

Основною метою цієї роботи є проектування та розробка веб-платформи для симуляції торгів криптоактивами, яка надає користувачам інтерактивне та безпечне середовище для вивчення фінансових ринків без ризику втрати реальних коштів. Система підтримує розмежування ролей користувача та адміністратора. Авторизацію реалізовано за допомогою протоколу JWT, з механізмом підтвердження електронної пошти для підвищення безпеки акаунтів. Платформа забезпечує моніторинг ринкових даних у реальному часі через WebSockets, можливість відкриття/закриття торгових позицій з налаштуванням захисних ордерів (Stop Loss, Take Profit), динамічний розрахунок прибутковості портфеля та візуалізацію фінансової аналітики за допомогою інтерактивних графіків.

У процесі виконання випускної роботи було розроблено програмний код із дотриманням сучасних стандартів та принципів інженерії програмного забезпечення (Clean Architecture, SOLID, модульність). Бекенд-частина побудована на фреймворку NestJS із використанням ORM Prisma для безпечної роботи з базою даних PostgreSQL. Для забезпечення портативності та стабільності середовища виконання застосовано технологію Docker. Клієнтська частина реалізована на базі Next.js (React), що забезпечує високу продуктивність та реактивність інтерфейсу.

Ключові слова: NESTJS, POSTGRES SQL, WEB, NEXT.JS, REACT, DOCKER, JWT, WEBSOCKETS, PRISMA, TRADING SIMULATION, FINTECH.

					ІПЗ – 121 – 25 - ІЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Web – застосунок для прогнозування ринкових трендів на основі ансамблів нейромереж Пояснювальна записка	Літ.	Арк.	Аркуші
Розроб.		О. О. ГАНЧЕВСЬКИЙ						
Керівник		Т. А. ВАКАЛЮК					3	83
Рецензент		Т. М. ЛОКТИКОВА				Житомирська політехніка, група ІПЗ-22-2		
Зав. каф.		Т. А. ВАКАЛЮК						

ЗМІСТ

ВСТУП	6
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1. Постановлення завдань, та їх вирішення	9
1.2. Аналіз схожого програмного забезпечення	11
1.3. Визначення архітектури програмного забезпечення	19
1.4. Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення.....	23
Висновки до першого розділу	27
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА WEB ДОДАТКУ ДЛЯ ПРОГНОЗУ РИНКОВИХ ТРЕНДІВ	29
2.1 Проектування бази даних.....	29
2.2 Розробка архітектури додатку	32
2.3 Розробка функціональних методів для роботи програми	36
Висновки до другого розділу.....	42
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК ВИКОРИСТАННЯ	46
ДОДАТКУ	46
3.1 Опис роботи з програмним додатком (Опис інтерфейсу).....	46
3.2 Роутинг та маршрутизація	50
3.3 Робота з авторизацією (JWT).....	54
3.4 User Stories та User Flows.....	57
3.5 Заходи безпеки бази даних	60
3.6 Застосування Postman для адміністрування та тестування API.....	64
Висновки до третього розділу	67

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ВИСНОВКИ.....	70
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	72
ДОДАТКИ.....	75

ВСТУП

Актуальність теми. У сучасних умовах стрімкого розвитку фінансових ринків та безпрецедентного зростання популярності інструментів для інвестування, зокрема цифрових активів, питання фінансової грамотності стає критично важливим для широких верств населення. Криптовалютний ринок характеризується надзвичайно високою волатильністю, цілодобовим режимом роботи та складними механізмами ціноутворення [1], що створює значні ризики для недосвідчених трейдерів. Традиційні методи навчання, засновані виключно на теоретичних матеріалах чи статичних даних, часто не забезпечують достатнього рівня підготовки, оскільки вони не здатні відтворити динаміку реальних торгів та психологічне навантаження, з яким стикається інвестор.

Спроби початківців одразу вийти на реальні біржі здебільшого призводять до втрати капіталу через брак практичного досвіду управління ризиками (Risk Management). Тому розробка спеціалізованих веб-платформ для симуляції торгівлі, які надають користувачам абсолютно безпечне середовище для тестування торгових стратегій, аналізу ринкових даних у реальному часі та навчання без ризику втрати реальних коштів, є надзвичайно актуальною задачею.

Крім того, з інженерної точки зору, розробка подібної платформи є складним викликом у галузі інженерії програмного забезпечення. Вона вимагає реалізації стійких архітектурних рішень для обробки потокових даних із мінімальною затримкою, забезпечення транзакційної цілісності бази даних під час конкурентних запитів та гарантування абсолютної точності фінансових обчислень. Відповідно, створення такого програмного продукту є актуальним як у контексті розвитку фінансових освітніх технологій (FinTech/EdTech), так і в площині проєктування високонавантажених веб-систем.

Мета роботи. Метою цієї роботи є проєктування та розробка вебзастосунку для прогнозування ринкових трендів на основі ансамблів нейромереж, інфраструктурною базою для якого виступає високопродуктивна платформа симуляції торгів криптоактивами («Trading Simulator Platform»). Таке комплексне рішення забезпечує безперервний збір актуальних ринкових даних для навчання

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		6

моделей, можливість виконання торгових операцій у віртуальному середовищі та інструменти для аналізу ефективності інвестиційного портфеля з урахуванням предиктивної аналітики.

Завдання дослідження. Для досягнення поставленої мети необхідно було вирішити комплекс взаємопов'язаних завдань. Насамперед вимагалось провести аналіз предметної області, існуючих методів прогнозування та специфіки роботи криптовалютних бірж для формування вимог до платформи, а також обґрунтувати вибір технологічного стеку для забезпечення високої продуктивності й математичної точності обчислень. Наступним етапом стало проєктування та нормалізація реляційної бази даних для надійного зберігання профілів користувачів, історії фінансових транзакцій і масивів ринкових даних, необхідних для валідації нейромережових алгоритмів. Після цього необхідно було розробити серверну частину системи (API) для обробки бізнес-логіки торгових ордерів, автентифікації користувачів і потокової трансляції котирувань через WebSocket-з'єднання, паралельно створюючи інтерактивний клієнтський інтерфейс (торговий термінал) з інтеграцією інструментів рендерингу фінансових даних у реальному часі. Насамкінець вимагалось налаштувати інфраструктуру для безпечного та ізольованого розгортання всіх компонентів екосистеми за допомогою технологій контейнеризації.

Об'єктом дослідження є процес інтелектуального аналізу динаміки ринку криптовалют та інформаційного забезпечення автоматизованих торгових операцій. У фокусі дослідження перебуває створення програмного комплексу, що забезпечує безперебійну взаємодію користувача з ринковими даними, механізмами виконання віртуальних угод та створює надійне середовище для тестування предиктивних трендових моделей

Предметом дослідження є методи та засоби розробки сучасного вебзастосунку для прогнозування ринкових трендів і симуляції торгівлі. У процесі реалізації використовуються такі технічні засоби: NestJS для серверної бізнес-логіки та інтеграції аналітичних модулів, React та Next.js для побудови

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ІЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		7

високоєфективного інтерфейсу користувача, PostgreSQL та Prisma для зберігання структурованої фінансової інформації й датасетів, а також Docker для контейнеризації середовища розробки та продуктивного розгортання. Такий технологічний підхід дозволяє створити масштабований, стабільний та зручний інструмент для вивчення механізмів фінансових ринків та апробації алгоритмів машинного навчання.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1. Постановлення завдань, та їх вирішення

Однією з ключових складових проєкту є ефективне проєктування бази даних, з акцентом на забезпечення цілісності, узгодженості та безпеки даних. Для реалізації цього завдання використовується реляційна система управління базами даних PostgreSQL у поєднанні з Prisma ORM [2, 3], що дозволяє організувати транзакційне виконання ордерів та захистити систему від стану гонитви (race conditions). Швидкість обробки запитів та підтримка великої кількості одночасних користувачів є критично важливими, зважаючи на те, що система має підтримувати відстеження прогресу, безперервне оновлення котирувань через WebSockets та розрахунок прибутків і збитків (PnL) в режимі реального часу.

Серверна частина веб-додатку реалізована за допомогою фреймворку NestJS [4], який забезпечує стабільну роботу, хорошу масштабованість і підтримку сучасних підходів до розробки, таких як REST API, модульність і залежності через ін'єкцію. Важливою архітектурною особливістю бекенду є наявність фонового процесу, який щосекунди аналізує поточні ціни активів в оперативній пам'яті та ініціює автоматичне закриття позицій при досягненні встановлених користувачем тригерів Stop Loss або Take Profit. Для гарантування абсолютної точності у фінансових обчисленнях та уникнення помилок роботи з плаваючою комою інтегровано бібліотеку decimal.js [5].

Клієнтська частина платформи побудована на базі React та Next.js з використанням TypeScript для забезпечення суворої типізації даних [6, 7, 8]. Інтерфейс користувача розроблено за допомогою Tailwind CSS [9] у стилістиці темної теми з елементами glassmorphism, що створює сучасний та комфортний для сприйняття візуальний досвід. Центральним елементом клієнтської логіки є торговий термінал, де за допомогою бібліотеки lightweight-charts реалізовано рендеринг графіків [10] японських свічок у реальному часі. Користувачі мають можливість змінювати таймфрейми, здійснювати торгові операції, використовувати режим малювання маркерами на графіку, а також отримувати

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		9

аналітичні прогнози від штучного інтелекту щодо цільової ціни активу. Для візуалізації стану портфеля розроблено інтерактивні кругові діаграми на базі recharts [11].

Для автентифікації та авторизації використовується JWT (JSON Web Token) [12, 13], що дозволяє реалізувати безпечний та масштабований механізм контролю доступу. З метою підвищення рівня захищеності облікових записів та фінансових даних в систему інтегровано механізм аутентифікації через надсилання коду підтвердження на пошту та безпечну процедуру відновлення пароля. Всі дані про користувачів та результати торгів зберігаються у захищеній базі даних із відповідним рівнем доступу, що гарантує конфіденційність інформації та надійну роботу симулятора торгів за будь-яких навантажень.

Як серверну частину обрано сучасний фреймворк NestJS, який використовує мову програмування TypeScript. Цей інструмент забезпечує розробку надійних, модульних та легко масштабованих веб-сервісів завдяки вбудованій підтримці ін'єкції залежностей (Dependency Injection) та чіткій архітектурі. Використання TypeScript на обох рівнях додатку (клієнтському та серверному) дозволяє гарантувати сувору типізацію даних і мінімізувати кількість помилок ще на етапі компіляції, що є критично важливим для фінансового симулятора.

Для реалізації клієнтської частини використано зв'язку бібліотеки React та фреймворку Next.js з використанням сучасної парадигми App Router. Це поєднання дозволяє створювати швидкий, оптимізований та інтерактивний інтерфейс користувача з гнучкими можливостями маршрутизації. Візуальне оформлення платформи реалізовано за допомогою фреймворку Tailwind CSS, що дало змогу швидко впровадити сучасну темну тему з ефектами glassmorphism (напівпрозорості та розмиття фону), забезпечуючи високу естетичність та ергономічність торгового терміналу.

Системою керування базами даних виступає PostgreSQL, яка надає широкі можливості для роботи з реляційними даними та гарантує високий рівень надійності. Для зручної, типізованої та безпечної взаємодії з базою даних

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				10
Змн.	Арк.	№ докум.	Підпис	Дата		

використовується Prisma ORM, що забезпечує транзакційне виконання ордерів і надійний захист від стану гонитви (race conditions) під час одночасних торгових операцій. Окрему увагу приділено точності математичних обчислень: для уникнення типових алгоритмічних проблем із числами з плаваючою комою у систему інтегровано спеціалізовану бібліотеку decimal.js. Вона гарантує абсолютну точність при розрахунках прибутків, збитків (PnL) та загального балансу користувачів.

Оскільки криптовалютний симулятор торгів вимагає миттєвої реакції на зміни ринку, для забезпечення швидкого двостороннього зв'язку між клієнтом та сервером використано технологію WebSockets за допомогою бібліотеки socket.io [14, 15]. Це дозволяє безперервно транслювати актуальні котирування активів без необхідності постійного перезавантаження сторінки. Візуалізація ринкових даних у торговому терміналі реалізована завдяки потужній бібліотеці lightweight-charts, яка ефективно рендерить графіки японських свічок та дозволяє користувачам взаємодіяти з ними. Для аналітичного відображення стану портфеля та візуалізації розподілу активів у профілі користувача застосовано бібліотеку recharts, що генерує інтерактивні кругові діаграми.

1.2. Аналіз схожого програмного забезпечення

Головним аналогом та ідейним натхненником для розроблюваної платформи є криптовалютна біржа Binance. Це один із найпопулярніших майданчиків, який надає користувачам доступ до торгівлі реальними активами, детальної статистики та управління портфелем. Інтерфейс торгового терміналу Binance (Рисунок 1.1) відрізняється високою інформативністю, підтримує оновлення даних у реальному часі через WebSockets та містить безліч інструментів для професійних трейдерів. Незважаючи на потужний функціонал, ця платформа має значний поріг входження для новачків. Велика кількість специфічних налаштувань, складні типи ордерів та перевантажений інтерфейс можуть заплутати недосвідченого користувача. Крім того, робота на реальній

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		11

біржі завжди пов'язана з фінансовими ризиками, що робить її непридатною для безпечного навчання та тестування торгових стратегій.

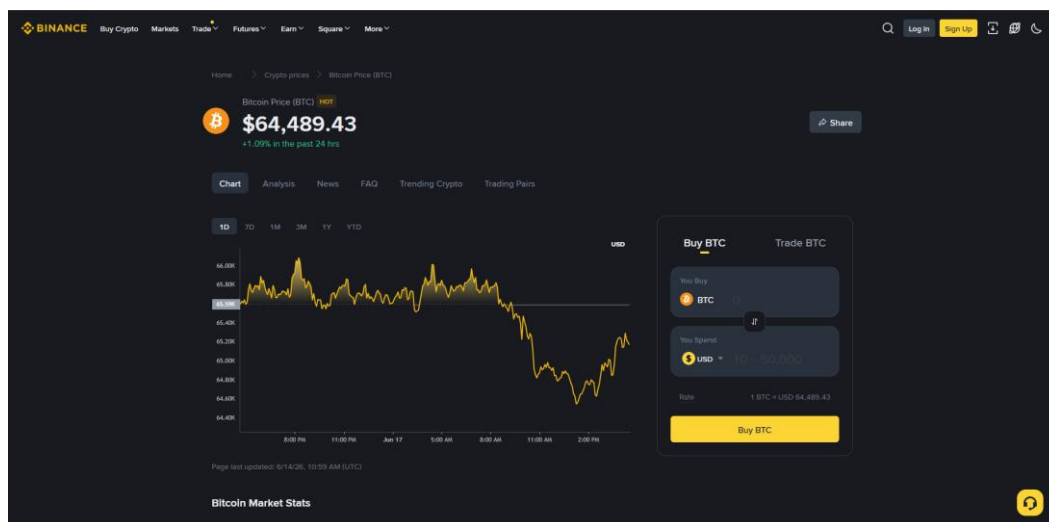


Рисунок 1.1 — Інтерфейс торгового терміналу Binance

Іншим важливим аналогом у сфері фінансової аналітики є платформа TradingView. Вона є світовим стандартом для побудови графіків та візуального технічного аналізу. Інтерфейс платформи (Рисунок 1.2) надає користувачам доступ до інтерактивних японських свічок, безлічі таймфреймів, інструментів малювання та індикаторів. Важливо зазначити, що у розроблюваному дипломному проєкті для клієнтської частини використовується бібліотека `lightweight-charts`, яка є спеціалізованою полегшеною версією графіків від розробників TradingView. Основною перевагою TradingView є неперевершена якість візуалізації даних. Однак, незважаючи на наявність базової функції симуляції торгів (Paper Trading), платформа в першу чергу орієнтована на глобальну аналітику і не пропонує ізольованого, сфокусованого середовища з вбудованими предиктивними AI-моделями для цілеспрямованого навчання.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				12
Змн.	Арк.	№ докум.	Підпис	Дата		

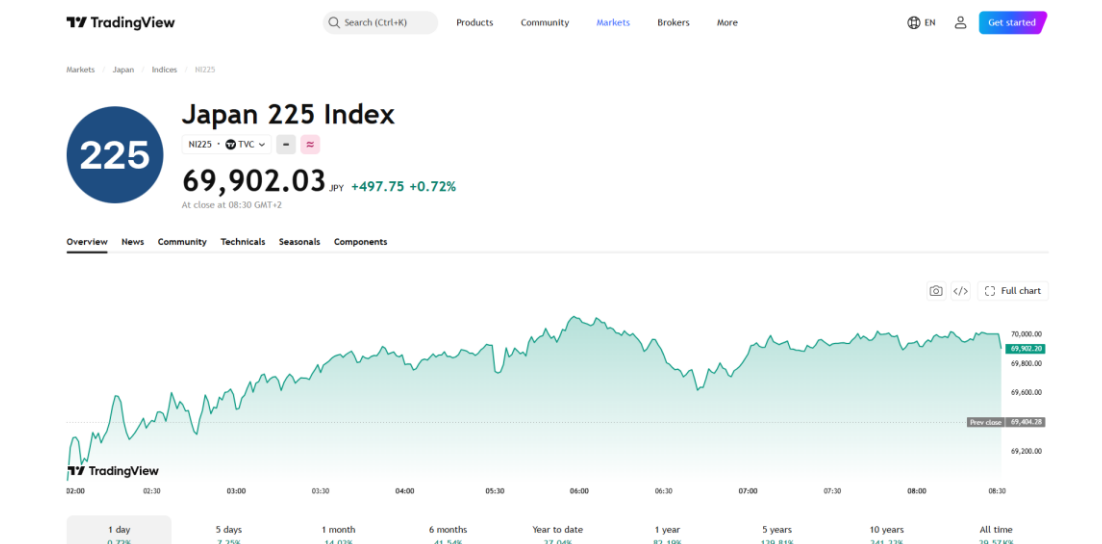


Рисунок 1.2 — Інтерфейс графіків TradingView

Аналізуючи ринок соціального трейдингу, варто також згадати платформу eToro, яка пропонує функціонал віртуального портфеля. Інтерфейс eToro (Рисунок 1.3) є значно простішим та більш дружнім до новачків, пропонуючи зручну візуалізацію активів. Проте ця система замкнута, не дозволяє глибоко налаштовувати торговий термінал під власні потреби і не має відкритого доступу до алгоритмічних інструментів управління ризиками, які б працювали на серверному рівні незалежно від відкритої вкладки браузера.

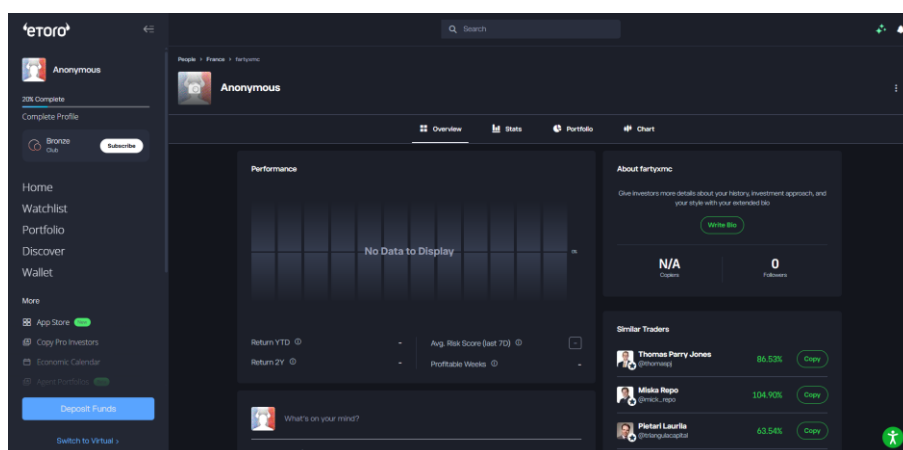


Рисунок 1.3 — Інтерфейс віртуального портфеля eToro

Розроблювана система "Trading Simulator Platform" покликана об'єднати найкращі риси проаналізованих аналогів, усунувши при цьому їхні ключові недоліки для цільової аудиторії. На відміну від Binance, наш додаток пропонує абсолютно безпечне середовище для симуляції торгів без реальних фінансових ризиків, повністю зберігаючи при цьому динаміку реального ринку завдяки

		О. О. ГАНЧЕВСЬКИЙ							Арк.
		Т. А. ВАКАЛЮК							
Змн.	Арк.	№ докум.	Підпис	Дата					13

інтеграції WebSockets для миттєвого оновлення котирувань. Порівняно з TradingView та eToro, наша платформа пропонує більш сучасний та оптимізований інтерфейс управління портфелем, де реалізовані інтерактивні кругові діаграми розподілу активів та динамічний розрахунок PnL. Головною ж відмінністю та перевагою є інтеграція предиктивної аналітики (AI Prediction Ensemble) безпосередньо в графік та надійне серверне ядро, яке за допомогою транзакційних черг та фонових процесів щосекунди контролює автоматичне закриття позицій за рівнями Stop Loss та Take Profit.

Криптовалютна біржа Binance є безперечним лідером на світовому ринку цифрових активів та виступає потужним еталоном для аналізу при проектуванні будь-яких торгових платформ чи симуляторів. Цей програмний продукт являє собою комплексну фінансову екосистему, яка об'єднує в собі спотову, маржинальну та ф'ючерсну торгівлю, а також інструменти для пасивного доходу, стейкінгу та поглибленої аналітики. Розглядаючи Binance з технічної та користувацької точок зору, варто відзначити високий рівень реалізації її торгового терміналу. Основний робочий простір трейдера побудований за модульним принципом, де центральне місце займає інтерактивний графік котирувань. Для забезпечення користувачів звичним набором інструментів технічного аналізу платформа інтегрує сторонні рішення, зокрема рушій від TradingView, що дозволяє накладати індикатори, малювати лінії тренду та використовувати різні таймфрейми. Навколо графіка органічно, хоча й досить щільно, розташовані стакан заявок (Order Book), історія угод ринку та багатофункціональна панель управління ордерами, яка підтримує ринкові, лімітні, стоп-лімітні та OCO (One-Cancels-the-Other) заявки.

З архітектурної точки зору платформа демонструє виняткові можливості обробки даних у режимі реального часу. Вона використовує оптимізовані з'єднання через WebSockets для миттєвої трансляції зміни цін, обсягів торгів та статусу виконання заявок без помітних затримок навіть при екстремальних пікових навантаженнях і високій волатильності ринку. Рушій зведення ордерів

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		14

(matching engine) здатен обробляти мільйони транзакцій на секунду, що є критично важливим показником для бірж такого масштабу. Управління портфелем на Binance реалізовано через розгалужену систему гаманців, яка дозволяє користувачеві детально відстежувати свій капітал, аналізувати прибутки та збитки (PnL) за різні періоди часу, а також переглядати розподіл активів. Технологічний стек біржі орієнтований на максимальну безпеку та відмовостійкість, що включає багаторівневу автентифікацію, суворе шифрування даних, використання холодних сховищ для активів та розподілені серверні потужності. Цей рівень інженерії є прикладом для наслідування при розробці серверного ядра нашого додатку, зокрема в питаннях транзакційного виконання ордерів через Prisma ORM та забезпечення абсолютної цілісності фінансових балансів користувачів за допомогою точних математичних обчислень. Аналіз цього сервісу дає чітке розуміння того, якими мають бути стандарти швидкодії та надійності зв'язку між клієнтом і сервером у сучасному веб-додатку фінансового спрямування.

Незважаючи на статус галузевого стандарту та колосальні технологічні можливості, платформа Binance має низку суттєвих недоліків, які роблять її неоптимальним вибором для користувачів, що прагнуть безпечно навчатися трейдингу, тестувати свої стратегії або просто знайомитися з ринком. Найбільш вираженою проблемою є надзвичайно високий поріг входження для новачків. Інтерфейс торгового терміналу та особистого кабінету критично перевантажений величезною кількістю функцій, показників, налаштувань та додаткових фінансових продуктів, які не стосуються безпосередньо процесу торгівлі. На екрані постійно присутні банери, спливаючі сповіщення та посилання на суміжні сервіси екосистеми. Це створює сильне когнітивне перевантаження, через що недосвідчений користувач витрачає більше часу на спроби зрозуміти навігацію платформи, ніж на вивчення базових ринкових механізмів. Візуальний стиль платформи, хоч і підтримує темну тему, є надто утилітарним і позбавленим

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				15
Змн.	Арк.	№ докум.	Підпис	Дата		

сучасних дизайнерських підходів, таких як мінімалізм чи візуальне фокусування на головному, що швидко призводить до втоми очей при тривалій роботі.

Крім того, робота на основній платформі Binance нерозривно пов'язана з реальними фінансовими та психологічними ризиками. Хоча біржа пропонує тестові мережі (testnets) для деяких своїх продуктів, наприклад для ф'ючерсів, вони вимагають складної окремої реєстрації, мають специфічний і менш інтуїтивний процес налаштування API ключів та не інтегровані безшовно в основний спотовий термінал. Таким чином, користувач не має можливості просто натиснути одну кнопку і перейти в режим безпечної симуляції у звичному інтерфейсі. Ще одним вагомим недоліком з точки зору освітнього процесу є повна відсутність вбудованих інструментів предиктивної аналітики чи підказок, які б допомагали користувачеві ухвалювати рішення на основі алгоритмів штучного інтелекту безпосередньо на графіку. Платформа надає лише "сирі" дані, залишаючи трейдера наодинці зі складною аналітикою. Також варто відзначити високу ресурсомісткість клієнтської частини: через велику кількість активних з'єднань, складну структуру об'єктної моделі документа (DOM) та безперервний рендеринг важких компонентів, інтерфейс біржі може працювати зі значними затримками на пристроях із середньою продуктивністю або при нестабільному інтернет-з'єднанні.

Сукупність цих факторів — перевантаженість інтерфейсу, відсутність зручного та безпечного симулятора в межах єдиної платформи, складність навігації та брак інтерактивних підказок на основі сучасних моделей штучного інтелекту — повністю обґрунтовує доцільність розробки спеціалізованої веб-платформи "Trading Simulator Platform". Проєктована система має на меті абстрагувати всю цю надлишкову технічну та візуальну складність. Вона надає користувачеві сфокусоване, стилістично приємне та безпечне середовище з сучасним дизайном на базі Tailwind CSS, де він зможе зосередитися виключно на розумінні поведінки графіків, управлінні віртуальним портфелем та налаштуванні алгоритмічних тригерів Stop Loss та Take Profit без ризику втрати власних коштів.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Binance — це професійна платформа для торгівлі цифровими активами, яка надає глибоку ліквідність та розширений інструментарій для технічного аналізу, що дає повний контроль над торговими операціями. Вона підходить для досвідчених трейдерів, інституційних інвесторів та алгоритмічних торгових систем. eToro, в свою чергу, є платформою соціального трейдингу, де користувачі можуть копіювати угоди успішних інвесторів без потреби в глибокому технічному аналізі, орієнтуючись на новачків та індивідуальних інвесторів.

Порівняння функцій Binance та eToro, наведене в таблиці, демонструє перевагу Binance у гнучкості налаштувань ордерів і розширеній аналітиці, тоді як eToro забезпечує простоту використання і мінімальні технічні бар'єри, що робить її ідеальним вибором для тих, хто шукає швидкий і зручний спосіб інвестування (табл. 1.1).

Таблиця 1.1

Порівняння ключових характеристик криптовалютних платформ Binance та eToro

Функція	Binance	eToro
Торговий термінал	Професійні графіки (TradingView), стакан заявок, безліч індикаторів	Спрощений інтерфейс, базові графіки, фокус на зручності
Кастомізація UI	Модульний робочий простір, темна/світла теми, налаштування сітки	Фіксований інтерфейс з обмеженими можливостями брендингу та зміни макета
Типи ордерів та автоматизація	Limit, Market, Stop-Limit, OCO, Trailing Stop, API для ботів	Прості Market/Limit, жорстко прив'язані Stop Loss та Take Profit

продовж. табл. 1.1

Соціальні та навчальні функції	Binance Academy (статті), таблиці лідерів (leaderboards)	CopyTrader (копіювання угод), соціальна стрічка новин, взаємодія
Управління портфелем	Детальний PnL, розподіл по гаманцях (спот, маржа, ф'ючерси)	Візуальний круговий графік, загальний баланс, просте відстеження
Мобільний доступ	Повнофункціональний додаток (перемикання Lite та Pro режимів)	Інтуїтивний мобільний додаток, адаптований виключно для початківців

Модель монетизації, існують різні моделі монетизації та витрат, пов'язані з використанням платформ для торгівлі, і кожна з них має свої особливості. Наприклад, Binance працює за моделлю комісій за кожну транзакцію (модель maker/taker), що робить її прозорою для активних користувачів. Однак, незважаючи на низькі базові ставки, використання Binance передбачає певні приховані складнощі для новачків. Зокрема, користувач повинен самостійно розраховувати комісії при використанні складних ордерів, маржинальної торгівлі або при виведенні коштів у різних блокчейн-мережах, що вимагає технічного розуміння процесу.

Інший підхід демонструє платформа eToro, яка працює переважно за моделлю спредів (різниця між ціною купівлі та продажу) замість прямих комісій за угоду. Завдяки цій системі вартість входу в позицію здається нульовою, що робить її фінансово привабливою для широкої аудиторії. До того ж, eToro пропонує функцію відкриття спотових позицій без кредитного плеча з нульовою

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		18

комісією на деякі активи, що мінімізує ризики для користувача. Для забезпечення прибутковості платформа має окрему пропозицію стягнення плати за неактивність та фіксовані комісії за виведення коштів, забезпечуючи зручність та гнучкість для компанії у монетизації пасивних рахунків.

Binance підходить для фінансових організацій і досвідчених трейдерів, де є потреба в інтеграції з API, використанні алгоритмічних ботів та доступі до складної аналітики. Вона надає гнучкість у налаштуванні та підтримує глибоку інтеграцію з іншими торговими інструментами. еТого, в свою чергу, орієнтований на індивідуальних інвесторів і новачків, які шукають швидкий доступ до ринку без технічних бар'єрів, що робить платформу зручною для тих, хто хоче зосередитися на копіюванні успішних стратегій, а не на технічному обслуговуванні власних позицій.

Отже, вибір Binance доцільний, якщо необхідна глибока ліквідність, відповідність вимогам професійного трейдингу, а також якщо ви готові вкладатися в вивчення технічних інструментів та підтримку власної торгової архітектури. У свою чергу, еТого є оптимальним варіантом для швидкого виходу на ринок, мінімальних технічних зусиль та орієнтації на широкий ринок інвесторів-початківців, яким важлива соціальна складова.

1.3. Визначення архітектури програмного забезпечення

Для розробки веб-додатку інтерактивної платформи симуляції торгів криптоактивами «Trading Simulator Platform» було обрано клієнт-серверну архітектуру, де клієнтська частина взаємодіє з сервером для обробки ринкових даних, збереження результатів торгівлі та забезпечення взаємодії з користувачем. Цей підхід забезпечує масштабованість і модульність системи, дозволяючи легко додавати нові функціональні можливості, такі як додаткові предиктивні AI-моделі або нові інструменти технічного аналізу, без значних змін у існуючій структурі (Рисунок 1.4).

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		19

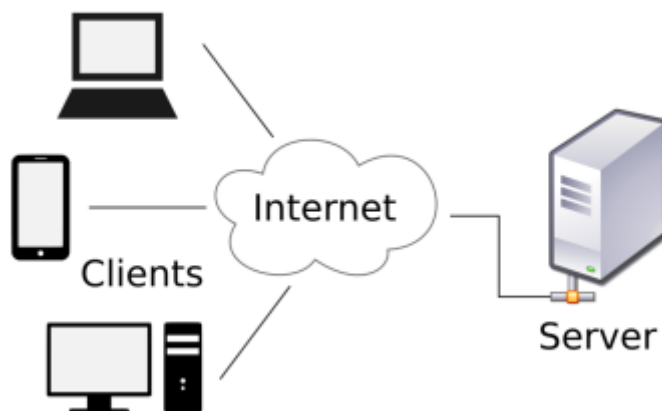


Рисунок 1.4 — Схематичне зображення клієнт-серверної архітектури платформи

Клієнтська частина платформи реалізована за допомогою сучасного фреймворку Next.js на базі бібліотеки React з використанням суворої типізації TypeScript. Використання TypeScript на клієнті забезпечує раннє виявлення помилок ще на етапі компіляції, автодоповнення коду в середовищі розробки та гарантує повну сумісність типів даних (end-to-end type safety) під час взаємодії із серверним API. Фронтенд відповідає за відображення головного торгового терміналу, який адаптовано під різні розміри екранів завдяки застосуванню сучасних підходів до стилізації (зокрема, utility-first CSS фреймворку Tailwind CSS).

До основних функцій клієнта належить рендеринг японських свічок у реальному часі за допомогою спеціалізованих бібліотек візуалізації (наприклад, Lightweight Charts), динамічна побудова кругових діаграм розподілу активів та миттєве оновлення балансу користувача. Для управління складним станом додатку застосовуються оптимізовані React-хуки, контексти та сучасні стейт-менеджери, що мінімізує зайві перемальовування компонентів [16]. Зокрема, імплементовано механізми мемоїзації (useMemo, useCallback) для ізоляції важких обчислень на стороні клієнта. Завдяки концепції побудови додатків із гібридним рендерингом (комбінація Server-Side Rendering для початкового завантаження та Client-Side Rendering для інтерактивності) забезпечуються високі показники продуктивності (Core Web Vitals).

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				20
Змн.	Арк.	№ докум.	Підпис	Дата		

Окремої уваги заслуговує ефективне управління станом через дуплексне WebSocket-з'єднання. Взаємодія з інтерфейсом відбувається безперервно, без необхідності повного перезавантаження сторінок. Клієнтська архітектура передбачає надійну обробку втрат з'єднання, автоматичне перепідключення до сокетів (exponential backoff) та кешування останніх отриманих даних. Використовується концепція оптимістичного оновлення (optimistic UI), коли дія користувача (наприклад, виставлення ордера) миттєво відображається в інтерфейсі ще до отримання підтвердження від сервера. Це дозволяє уникнути візуальних затримок при відображенні котирувань, що критично важливо для моніторингу високоволатильних ринків, і суттєво покращує загальний користувацький досвід трейдера.

Серверна частина системи побудована на базі модульного фреймворку NestJS, що забезпечує стабільну, чисту та масштабовану структуру коду за рахунок активного використання патерну впровадження залежностей (Dependency Injection). Кожен логічний домен системи — авторизація, управління балансом, маршрутизація новин, валідація даних чи обробка ордерів — виділений в окремий незалежний модуль, що значно спрощує модульне тестування та подальшу підтримку. На рівні обробки вхідних запитів реалізовано строгу валідацію через Data Transfer Objects (DTO) з використанням class-validator, що відсікає некоректні дані ще до етапу виконання бізнес-логіки.

Робота з реляційною базою даних PostgreSQL реалізована через сучасне середовище Prisma ORM. Такий підхід дозволяє автоматизувати створення міграцій, забезпечує типізований доступ до даних та гарантує надійність збереження конфіденційних профілів користувачів, відкритих позицій та історії угод. Для оптимізації швидкодії БД налаштовано пулінг з'єднань (connection pooling) та створено складені індекси для таблиць із фінансовими транзакціями, що прискорює вибірку великих масивів історичних даних.

Особливістю серверної архітектури є інтеграція спеціалізованого шару обробки подій у реальному часі за допомогою WebSockets та автономного

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		21

фонового воркера (Background Worker). Цей воркер функціонує незалежно від основного потоку обробки HTTP-запитів, щосекунди звіряючи поточні ціни в оперативній пам'яті (або в in-memory сховищі, такому як Redis) з лімітами Stop Loss та Take Profit для автоматичного закриття угод без блокування Event Loop платформи Node.js [17].

Для запобігання стану гонитви (race conditions) під час одночасного виконання ордерів (наприклад, коли кілька запитів намагаються списати кошти з одного балансу) застосовано механізм ізольованих транзакцій баз даних з відповідними рівнями блокування (row-level locks), які забороняють модифікацію балансу іншими процесами до повного завершення поточної фінансової операції. Для захисту доступу та маршрутів API впроваджено механізм автентифікації на основі токенів JWT (з поділом на короткострокові Access та довгострокові Refresh токени) разом із можливостями розширення до двофакторної автентифікації. Додатково серверний рівень захищений механізмами обмеження частоти запитів (Rate Limiting) для протидії DDoS-атакам.

Усі фінансові обчислення виконуються через виділений математичний модуль на базі бібліотеки decimal.js. Це ізолює логіку розрахунку капіталу, маржі та прибутку від архітектурних обмежень обробки чисел із плаваючою комою (Floating-Point Arithmetic) у середовищі JavaScript. Використання цієї бібліотеки повністю усуває проблему втрати математичної точності, що є безкомпромісною та жорсткою вимогою для будь-яких FinTech-продуктів.

Завдяки чіткому розподілу обов'язків (Separation of Concerns) між фронтендом і бекендом, а також модульній структурі NestJS та компонентному підходу Next.js, архітектура системи демонструє високий потенціал до масштабування. Проєкт побудований за принципом «Модульного Моноліту», який за потреби легко декомпозиється на мікросервіси. Використання єдиного стандарту REST API [18] та WebSocket-шлюзів гарантує, що клієнт і сервер можуть оновлюватися та розгортатися незалежно один від одного. Це дозволяє розробникам ізольовано модернізувати окремі сервіси, наприклад, інтегрувати

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				22
Змн.	Арк.	№ докум.	Підпис	Дата		

нові платіжні шлюзи, підключати зовнішні джерела постачання ліквідності (через FIX-протокол або сторонні API) або розширювати аналітичний AI-модуль.

Інтеграція предиктивної аналітики (зокрема, з використанням TensorFlow.js для побудови ансамблю моделей прогнозування) [19] органічно вписується в існуючу архітектуру без порушення стабільності основного торгового ядра. Додатково, подібна архітектура ідеально адаптована для контейнеризації за допомогою Docker. Налаштування ізольованих контейнерів для клієнта, сервера та бази даних створює консистентне середовище як для локальної розробки, так і для швидкого розгортання в сучасних хмарних PaaS-середовищах. Впровадження конвеєрів безперервної інтеграції та розгортання (CI/CD) забезпечує автоматизоване тестування кожної зміни коду, гарантуючи загальну відмовостійкість, мінімальну затримку обробки подій та високу рентабельність довгострокової підтримки програмного продукту.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення

Для розробки веб-додатку симулятора криптовалютних торгів було обрано інструментальні засоби, які відповідають найвищим вимогам щодо продуктивності, масштабованості, точності обчислень та зручності в довгостроковому обслуговуванні. Вибір технологічного стека був зумовлений необхідністю забезпечити безперервну трансляцію ринкових даних у реальному часі, транзакційну безпеку фінансових операцій та ефективне управління портфелем користувача. У контексті симулятора біржі, де ринкові умови змінюються щомілісекунди, критично важливо звести до мінімуму мережеву затримку (latency) та гарантувати високу відмовостійкість архітектури під час пікових навантажень або різких коливань цін.

Для серверної частини (бекенду) обрано фреймворк NestJS, оскільки він базується на строго типізованій мові TypeScript і "під капотом" використовує оптимізовані HTTP-сервери (Express або Fastify). NestJS підтримує сучасні архітектурні підходи, такі як глибока модульність, предметно-орієнтоване

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				23
Змн.	Арк.	№ докум.	Підпис	Дата		

проектування (DDD) [20] та ін'єкція залежностей (Dependency Injection). Використання вбудованих механізмів (Guards, Interceptors, Pipes) дозволяє реалізувати надійний шар валідації вхідних даних та безпечної авторизації запитів. Інтеграція WebSockets на базі бібліотек Socket.io або ws дозволяє реалізувати стабільний повнодуплексний двосторонній зв'язок [21] для миттєвого оновлення котирувань без накладних витрат, притаманних класичним HTTP-запитам. Для роботи з базою даних використовується PostgreSQL у поєднанні з сучасним інструментом Prisma ORM. Prisma не лише забезпечує зручне управління міграціями, але й автоматично генерує типи доступу до даних, що повністю виключає ризик виникнення помилок невідповідності структур даних (Type Mismatch) на етапі компіляції. Задля абсолютної точності фінансових розрахунків на рівні ядра системи інтегровано математичну бібліотеку decimal.js. У криптовалютах, де обсяг активу може мати до 18 знаків після коми (наприклад, Wei або Satoshi), стандартний формат чисел із плаваючою комою IEEE 754 неминуче призводить до втрати точності. decimal.js вирішує цю проблему, обробляючи числа як рядки та гарантуючи точність будь-яких арифметичних операцій при розрахунку комісій чи маржі.

Клієнтська частина побудована на базі фреймворку Next.js та компонентної бібліотеки React, що дозволяє створювати швидкі, оптимізовані та інтуїтивно зрозумілі інтерфейси користувача. Завдяки сучасній архітектурі App Router, система гнучко поєднує рендеринг на стороні сервера (SSR) для миттєвого первинного завантаження та пошукової оптимізації, а також рендеринг на стороні клієнта (CSR) для динамічних торгових віджетів. Візуалізація складних фінансових даних реалізована за допомогою вузькоспеціалізованих рішень, що враховують обмеження ресурсів браузера. Для високопродуктивного рендерингу графіків японських свічок інтегровано бібліотеку lightweight-charts, яка використовує технологію HTML5 Canvas. Це дозволяє системі плавно відмальовувати десятки тисяч точок даних із частотою 60 кадрів на секунду, не перевантажуючи центральний процесор користувача. Водночас для побудови

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		24

інтерактивних кругових діаграм розподілу активів та аналітики використовується бібліотека recharts, що базується на векторній графіці SVG і забезпечує чіткість зображення на дисплеях високої роздільної здатності (Retina).

Порівняльний аналіз систем управління базами даних доводить беззаперечні переваги PostgreSQL для використання у фінансовому проєкті порівняно з такими системами, як MongoDB чи MariaDB. PostgreSQL підтримує механізм багатоверсійного управління конкурентним доступом (MVCC), що дозволяє сотням користувачів одночасно зчитувати актуальні котирування та баланси, не блокуючи таблиці для запису нових угод. На відміну від нереляційних рішень на кшталт MongoDB, які орієнтовані на кінцеву узгодженість (Eventual Consistency), PostgreSQL гарантує строгу ACID-відповідність (атомарність, узгодженість, ізольованість, довговічність) [27]. Це є критичною вимогою для торгової платформи, оскільки процеси списання фіатних коштів та нарахування криптовалюти повинні виконуватися як єдина неподільна транзакція, відкидаючи будь-які зміни у разі системного збою. У порівнянні з MariaDB, PostgreSQL пропонує значно ширші можливості для роботи з неструктурованими даними формату JSONB, що дозволяє гнучко зберігати специфічні метадані ордерів без ускладнення та денормалізації основної реляційної схеми бази даних (табл. 1.2).

Таблиця 1.2

Порівняльний аналіз технічних характеристик систем управління базами даних

Критерій	MongoDB	PostgreSQL	MariaDB
Тип СУБД	Документна (NoSQL), зберігає JSON-документи	Реляційна (SQL), підтримує JSONB для неструктурованих даних	Реляційна (SQL), підтримка JSON для зберігання

продовж. табл. 1.2

Масштабованість	Горизонтальна (для неструктурованих даних)	Горизонтальна і вертикальна, підтримка шардінгу	Вертикальна, з можливістю шардінгу
Продуктивність на складних запитах	Висока для неструктурованих даних	Висока для аналітичних та транзакційних запитів	Висока для CRUD-операцій
Робота з великими обсягами даних	Оптимізована для великих неструктурованих обсягів	Підтримує великі обсяги структурованих фінансових даних	Добре масштабується при помірних навантаженнях
Безпека даних	Додаткова конфігурація для безпеки	Розширене шифрування, аудит та сувора консистентність (ACID)	Високий рівень безпеки, підтримка доступу
Резервне копіювання та відновлення	Потребує зовнішніх інструментів	Вбудоване резервне копіювання і швидка реплікація	Підтримка бекапів та реплікації

Вибір PostgreSQL для проєкту був обумовлений його здатністю ефективно обробляти складні запити, підтримувати великі обсяги структурованих даних та забезпечувати найвищу надійність для транзакційних операцій. Це робить його ідеальним вибором для платформи симуляції торгів, де критично важливо безпечно зберігати дані користувачів, історію угод, статуси активних позицій та безпомилково розраховувати поточний прогрес портфелю.

Висновки до першого розділу

У першому розділі було розглянуто загальну концепцію, цільове призначення та функціональні особливості розроблюваної системи. Основна ідея полягає у створенні інтерактивної платформи «Trading Simulator Platform», яка поєднує функціонал повноцінного криптовалютного торгового термінала з інструментами портфельного менеджменту та предиктивної аналітики. Такий підхід дозволяє задовольнити потреби користувачів-початківців у безпечному середовищі для навчання та тестування торгових стратегій без фінансових ризиків, одночасно надаючи досвідченим трейдерам зручний інтерфейс для моніторингу ринку та відстеження прибутків і збитків (PnL) у реальному часі.

З технічної точки зору, система базується на сучасній та оптимізованій архітектурі. Фронтенд реалізовано у вигляді динамічного додатка на базі React та Next.js із використанням TypeScript та Tailwind CSS для забезпечення швидкого і зручного інтерфейсу з елементами glassmorphism. Для безперервного відображення котирувань інтегровано бібліотеку lightweight-charts, а для аналітики портфеля — recharts. Серверна частина побудована на модульному фреймворку NestJS, що гарантує надійну обробку бізнес-логіки. Взаємодія між клієнтом та сервером у режимі реального часу відбувається через WebSockets (socket.io), а для безпечного збереження структурованих даних, історії угод та балансів обрано PostgreSQL у поєднанні з транзакційною безпекою Prisma ORM.

Особливу увагу в архітектурі приділено безпеці, точності обчислень та автоматизації процесів. Аутентифікація користувачів реалізована через JSON Web Tokens (JWT), що доповнюється механізмами двофакторної аутентифікації

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				27
Змн.	Арк.	№ докум.	Підпис	Дата		

(2FA) для максимального захисту облікових записів. Для уникнення похибок, пов'язаних із числами з плаваючою комою, на серверному рівні інтегровано бібліотеку decimal.js, яка гарантує абсолютну математичну точність фінансових транзакцій. Крім того, на рівні серверного ядра реалізовано автономні фонові процеси, які безперервно відстежують зміни цін і автоматично виконують ордери за тригерами Stop Loss та Take Profit, що робить систему стійкою, надійною та автономною.

У підсумку, платформа вдало поєднує переваги провідних рішень на ринку: вона пропонує глибину та професійність інструментів Binance, якість візуалізації TradingView та простоту портфельного управління eToro. При цьому система повністю усуває реальні фінансові ризики, знижує поріг входження завдяки інтуїтивному інтерфейсу та пропонує унікальні можливості навчання за рахунок інтеграції AI-моделей для прогнозування цін. Завдяки цій комбінації характеристик розроблена система має значний потенціал стати ефективним освітнім та аналітичним інструментом у сфері сучасних фінансових технологій.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				28
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА WEB ДОДАТКУ ДЛЯ ПРОГНОЗУ РИНКОВИХ ТРЕНДІВ

2.1 Проектування бази даних

Проектування реляційної бази даних для веб-додатку симулятора криптовалютних торгів включає кілька послідовних етапів, які відображають структуру, семантику та взаємозв'язки сутностей (рис. 2.1). На першому етапі проводиться глибокий аналіз вимог до предметної області системи, де визначаються основні об'єкти, з якими буде взаємодіяти платформа, та логіка зв'язків між ними. На цьому етапі кожному об'єкту призначаються атрибути; наприклад, у сутності торгової позиції формуються поля ідентифікатора, символу активу, кредитного плеча, обсягу, ціни входу та детермінованих лімітних рівнів закриття угоди. Після цього визначаються первинні ключі для кожної таблиці у криптографічно безпечному форматі UUID (версії 4). Використання UUID замість класичних автоінкрементних ідентифікаторів (Integer) повністю унеможливорює атаки типу перебору (enumeration attacks), приховуючи від користувачів загальну кількість реєстрацій чи угод у системі. Наступним кроком є нормалізація бази даних до третьої нормальної форми (3NF) для мінімізації надмірності даних, усунення аномалій оновлення та підтримки повної цілісності. Це дозволяє логічно розбити великі масиви інформації на ізольовані сутності. Далі відбувається декларативна розробка схеми за допомогою інструментарію Prisma ORM для ефективної вибірки, оновлення та видалення даних [3]. На фінальних етапах проводиться оптимізація: налаштування B-Tree індексів для полів, за якими здійснюється частий пошук (наприклад, електронна пошта або символ активу), тестування на відсутність конфліктів у даних, після чого готова архітектура впроваджується в роботу з подальшим моніторингом повільних запитів.

У процесі проектування була розроблена база даних, що налічує вісім взаємопов'язаних таблиць. Система побудована на ключових сутностях: User (Користувач), Wallet (Гаманець), TradePosition (Торгова позиція), Transaction (Транзакція), TrackedSymbol (Активи, що відслідковуються) та Announcement

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				29
Змн.	Арк.	№ докум.	Підпис	Дата		

(Новини). Така конфігурація забезпечує мінімально необхідну, але повністю функціональну та масштабовану архітектуру для платформи симуляції торгів. Відмова від надмірно розгалужених структур та жорстке прив'язування активних угод до конкретного користувача спрощує об'єктно-реляційне відображення і гарантує миттєвий розрахунок поточних балансів. Особлива увага приділена таблиці Transaction, яка реалізує архітектурний патерн "Append-Only Ledger" [23, 24] (реєстр тільки для додавання). Це означає, що система може виконувати CRUD-операції над профілями чи активними ордерами, але історія фінансових транзакцій є незмінною (immutable) — записи до неї лише додаються і ніколи не видаляються чи модифікуються, що гарантує суворий аудиторський слід.

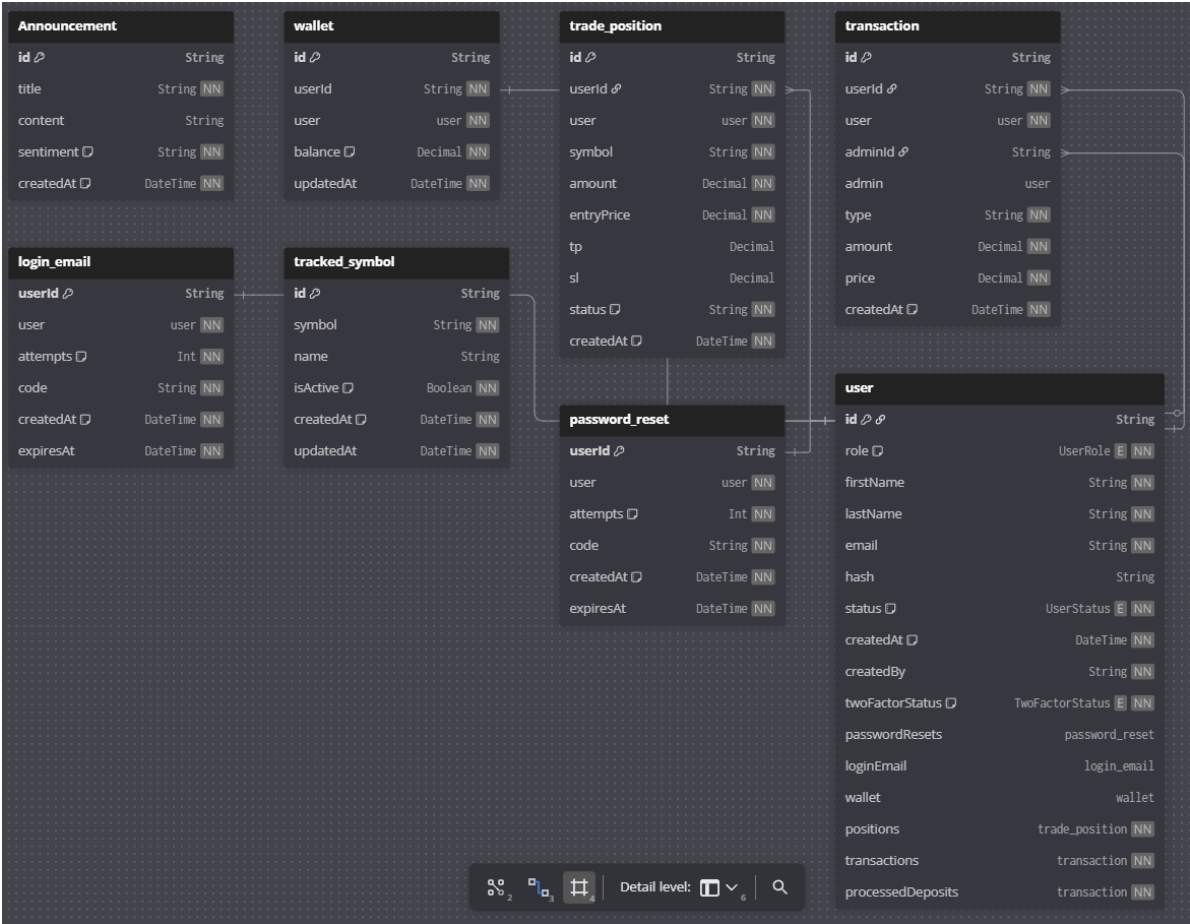


Рисунок 2.1 — Діаграма сутностей та зв'язків

Для контролю доступу в системі застосовано патерн управління доступом на основі ролей (RBAC). Ролі зберігаються безпосередньо в полі таблиці User за допомогою переліку (enum) UserRole і перевіряються в кожному HTTP-запиті на сервер за допомогою криптографічного підпису JWT-токена. У системі

передбачено дві основні ролі. Роль адміністратора (ADMIN) надає розширений доступ до управління платформою, дозволяє створювати глобальні новини системи та ініціювати транзакції для нарахування віртуального депозиту користувачам. Звичайний користувач (USER) отримує ізольований простір для здійснення торгових операцій, встановлення ордерів та перегляду власної статистики. Додатковий рівень безпеки забезпечується хешуванням паролів за допомогою сучасного алгоритму Argon2, а також інтеграцією таблиць LoginEmail та PasswordReset для контролю спроб входу та захисту від атак методом грубої сили (brute-force). Поле TwoFactorStatus дозволяє гнучко управляти налаштуваннями двофакторної автентифікації на рівні бази даних, тимчасово блокуючи доступ до гаманця у разі підозрілої активності.

У моделі даних реалізовано кілька суворих типів зв'язків із налаштуванням каскадної поведінки для забезпечення безперебійної торгової логіки. Зв'язок "один-до-одного" (1:1) існує між сутностями User та Wallet, оскільки кожен трейдер повинен мати єдину точку збереження загального балансу. Зв'язок "один-до-багатьох" (1:N) встановлено між користувачем та його торговими позиціями (TradePosition), що дозволяє системі швидко агрегувати дані за всіма відкритими угодами для розрахунку маржі. Найважливішим архітектурним рішенням при проектуванні бази даних стало використання типу Decimal із фіксованою точністю (18, 8) для всіх фінансових полів: баланс, обсяг позиції, ціна входу, рівні Stop Loss та Take Profit. Формат (18, 8) означає загальну довжину числа у 18 знаків, 8 з яких відведено під дробову частину, що є загальноприйнятим стандартом для криптовалют (відповідає розмірності 1 Satoshi для Bitcoin). Використання рушія PostgreSQL разом із цим типом даних повністю унеможлиблює проблему втрати точності під час агрегації, притаманну базовим типам Float чи Double Precision (стандарт IEEE 754), що є критичною вимогою для розробки стійких FinTech-платформ. Додатково, для оновлення фінансових зв'язків застосовуються механізми блокування на рівні рядків (Row-Level

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		31

Locking), які ізолюють баланс гаманця під час транзакції, виключаючи будь-які паралельні перезаписи.

2.2 Розробка архітектури додатку

В межах розробки та проєктування архітектури веб-платформи «Trading Simulator Platform» першочерговим завданням є забезпечення надійної ізоляції, високої портативності та швидкого розгортання всіх компонентів системи. Для ефективного вирішення цієї задачі було обрано сучасну систему контейнеризації Docker [25, 26], яка надає можливість інкапсулювати кожен окремий мікросервіс платформи у незалежні та ізольовані контейнери. Такий підхід повністю мінімізує ризики виникнення конфліктів залежностей між середовищем локальної розробки та продуктивним сервером, суттєво підвищує загальний рівень безпеки даних, а також спрощує подальше горизонтальне масштабування окремих вузлів системи при зростанні навантаження. Керування життєвим циклом, взаємозв'язками, внутрішніми віртуальними мережами та спільними ресурсами цих контейнерів реалізовано за допомогою інструменту оркестрації Docker Compose. Спеціалізований файл конфігурації `docker-compose.yml` детально описує параметри запуску кожного сервісу, правила їхнього автоматичного перезапуску, внутрішні ізольовані мережі, постійні сховища даних на хост-машині та змінні середовища, необхідні для коректної ініціалізації додатку.

Серверний сервіс API, який у структурі проєкту розробляється на базі модульного фреймворку NestJS та мови TypeScript, функціонує всередині окремого ізольованого Docker-контейнера під керуванням середовища виконання Node.js. Цей контейнер інтегрований у спеціально створену приватну віртуальну мережу типу bridge, що забезпечує надійну ізоляцію ядра бізнес-логіки платформи від безпосереднього зовнішнього доступу з глобальної мережі. Серверний сервіс взаємодіє з реляційною базою даних та іншими внутрішніми компонентами виключно в межах цього захищеного мережевого периметра. Конфігурація контейнера передбачає використання оптимізованих багатоетапних збірок (multi-stage builds), що дозволяє відокремити процес компіляції TypeScript

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				32
Змн.	Арк.	№ докум.	Підпис	Дата		

від фінального образу, суттєво зменшити його розмір та повністю виключити інструменти розробника з продуктивного середовища, підвищуючи стійкість торгової платформи до потенційних зовнішніх загроз.

Клієнтський сервіс користувацького інтерфейсу базується на технологічному стеку React та Next.js із використанням архітектури App Router. У контейнеризованому середовищі розробки фронтенд-сервіс запускається за допомогою відповідних менеджерів пакетів та скриптів автоматизації, що забезпечує підтримку функціоналу гарячого перезавантаження інтерфейсу (Hot Module Replacement). Для гарантування коректного та стабільного відстеження змін у вихідному коді під час монтування локальних директорій у робочу область контейнера, у конфігурації прописано спеціальні змінні середовища, зокрема CHOKIDAR_USEPOLLLING. За замовчуванням бібліотека Chokidar, інтегрована у внутрішні інструменти Next.js для моніторингу файлової системи, використовує нативні асинхронні механізми операційної системи, такі як inotify на ядрах Linux або fsevents у середовищі macOS, які забезпечують мінімальну затримку та економне використання системних ресурсів. Проте в ізольованих контейнерах Docker, які працюють на хост-системах під керуванням Windows або у специфічних віртуальних середовищах, ці системні події можуть транслюватися некоректно або повністю ігноруватися. Активація режиму опитування файлів (polling) у конфігурації розгортання гарантує безперебійну роботу інструментів розробника та миттєве оновлення торгового терміналу при модифікації коду користувачем незалежно від архітектури базової ОС.

Центральним елементом розподілу, балансування та безпеки мережевого трафіку в архітектурі системи виступає високопродуктивний веб-сервер Nginx, який функціонує як зворотний проксі-сервер (reverse proxy) [27]. Він виступає єдиною точкою входу для всіх зовнішніх HTTP та WebSocket запитів, що надходять від клієнтських додатків, і забезпечує їхню інтелектуальну маршрутизацію на відповідні внутрішні порти та сервіси у Docker-мережі. Роль Nginx є критично важливою для забезпечення стабільного функціонування

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				33
Змн.	Арк.	№ докум.	Підпис	Дата		

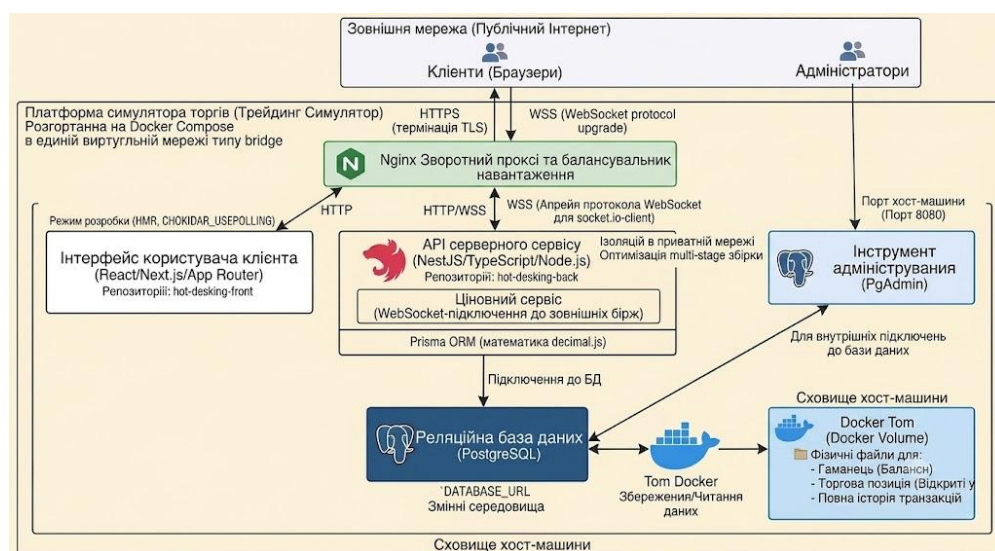
«Trading Simulator Platform», оскільки окрім стандартного проксіювання запитів до REST API та статичних ресурсів Next.js, він конфігурується на підтримку оновлення протоколу (Protocol Upgrade) для WebSocket-з'єднань socket.io-client. Це є першочерговою умовою для безперешкодної, низькозатримкової трансляції потоків ринкових котирувань і японських свічок у реальному часі без втрати пакетів. Сервер Nginx додатково налаштований на термінацію SSL/TLS з'єднань, що забезпечує надійне наскрізне шифрування даних користувачів, та має сувору політику автоматичного перезапуску при виявленні збоїв.

Для організації збереження та обробки структурованої інформації в архітектурі виділено контейнер на базі офіційного образу реляційної системи управління базами даних PostgreSQL. Усі чутливі параметри підключення до СКБД, включаючи унікальний рядок автентифікації DATABASE_URL, який використовується інструментарієм Prisma ORM для генерації клієнтських запитів та виконання міграцій, передаються в систему безпечним чином через змінні середовища Docker Compose. Щоб повністю унеможливити втрату фінансових даних користувачів, поточного стану балансів гаманців (Wallet), відкритих угод (TradePosition) та повної історії транзакцій при плановому перезапуску або аварійному завершенні роботи контейнерів, у конфігурації реалізовано механізм постійних сховищ даних (Docker Volumes). Це дозволяє винести фізичні файли таблиць бази даних за межі життєвого циклу контейнера на постійний накопичувач хост-машини, гарантуючи високу відмовостійкість та персистентність інформаційного шару торгового ядра.

Для спрощення процесів адміністрування, моніторингу загальної продуктивності та оперативного керування структурами даних у реляційній базі, до загальної інфраструктури додатку інтегровано контейнер спеціалізованого інструменту PgAdmin. Він надає адміністраторам платформи зручний та графічно насичений веб-інтерфейс для взаємодії з розгорнутою СКБД PostgreSQL у межах спільної віртуальної мережі додатку. PgAdmin сконфігуровано на автоматичне підключення до контейнера бази даних, що надає можливість безпечного

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		34

виконання налагоджувальних SQL-запитів, детального аналізу логів, ручного аудиту фінансових транзакцій та оперативного контролю цілісності реляційних зв'язків. Веб-інтерфейс цього сервісу виведено на окремий виділений порт хост-машини, наприклад 8080, що дозволяє надійно ізолювати адміністративні панелі керування від публічних маршрутів і користувацького трафіку основного додатку.



		О. О. ГАНЧЕВСЬКИЙ			ІІЗ – 121 – 25 - ІЗ	Арк.
		Т. А. ВАКАЛЮК				35
Змн.	Арк.	№ докум.	Підпис	Дата		

2.3 Розробка функціональних методів для роботи програми

У проєкті розробка функціональних методів базується на сучасній модульній архітектурі серверного фреймворку NestJS, який активно застосовує патерн впровадження залежностей (Dependency Injection) для ефективного управління компонентами системи [28]. Головним вузлом, що концентрує бізнес-логіку платформи симуляції торгів, є сервіс TradeService. Цей клас відповідає за обробку торгових операцій користувачів, обчислення фінансових показників портфеля та безпечну взаємодію з реляційною базою даних. У реалізації відповідного функціоналу використовуються основні залежності, які впроваджуються через конструктор сервісу. Зокрема, PrismaService забезпечує доступ до бази даних та надає можливість використання транзакцій, тоді як PriceService виступає внутрішнім постачальником актуальних ринкових котирувань, які безперервно оновлюються в оперативній пам'яті через WebSocket-з'єднання з зовнішніми біржами.

Одним із ключових функціональних методів системи є executeOrder, який слугує єдиним публічним фасадом для маршрутизації операцій купівлі та продажу криптоактивів. Цей метод приймає об'єкт передачі даних (DTO), що містить ідентифікатор користувача, символ активу, обсяг угоди, напрямок торгівлі, а також опціональні рівні автоматичного закриття позиції: Stop Loss (обмеження збитків) та Take Profit (фіксація прибутку). У методі обробки запиту спочатку виконується сувора валідація вхідних параметрів та перевірка наявності користувача в системі. Далі сервіс звертається до методу getLatestPrice постачальника даних, щоб миттєво отримати актуальну вартість обраного активу. Це архітектурне рішення гарантує, що угода буде виконана за найбільш точною ринковою ціною на момент запиту, виключаючи мережеві затримки, які могли б виникнути при прямому запиті до стороннього API під час кожної транзакції.

Після отримання актуальної ціни система переходить до математичних обчислень. Критично важливою особливістю розробки функціональних методів є використання спеціалізованої бібліотеки decimal.js для розрахунку загальної

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				36
Змн.	Арк.	№ докум.	Підпис	Дата		

вартості ордера та перевірки достатності коштів на віртуальному гаманці користувача. Застосування цього інструменту гарантує абсолютну точність обчислень і повністю захищає платформу від класичних похибок роботи з числами з плаваючою комою, що є абсолютно неприпустимим у фінансових симуляторах. Якщо перевірка балансу проходить успішно, логіка перенаправляється до ізольованих методів processBuy або processSell, залежно від типу ордера.

Для забезпечення цілісності даних усі подальші зміни виконуються в межах єдиної транзакції бази даних за допомогою методу Prisma.\$transaction. Використання шаблону транзакцій (Unit of Work, реалізованого силами ORM) гарантує атомарність операцій: списання коштів з гаманця, створення нового запису відкритої угоди в таблиці TradePosition та фіксація події в історії транзакцій об'єднуються в один блок. Якщо на будь-якому етапі виникає помилка — наприклад, стан гонитви (race condition) при одночасному надсиланні кількох запитів — усі внесені зміни автоматично відкочуються. Це гарантує, що баланс користувача ніколи не буде змінено без фактичного створення відповідної позиції. Завдяки цьому система залишається консистентною та надійною.

Крім базової логіки виконання ордерів, життєво важливим функціональним алгоритмом є метод checkAndTriggerPriceOrders, який забезпечує автоматичне закриття позицій. У методі життєвого циклу onModuleInit() ініціюється автономний фоновий процес (воркер), який щосекунди викликає цей алгоритм. Він збирає всі відкриті позиції з бази даних, порівнює їхні цільові рівні з поточними котируваннями і при настанні тригерних умов автоматично ініціює операцію продажу. Особливу увагу в цьому методі приділено обробці помилок "холодного старту": система перевіряє повноту завантажених котирувань, щоб запобігти хибному закриттю угод у момент, коли WebSocket-з'єднання ще не встигло передати перші ціни.

Додатково реалізовано метод getPortfolio, який агрегує фінансові дані користувача, динамічно вираховуючи нереалізований прибуток чи збиток (PnL)

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		37

та загальний капітал. Для цього дані з бази PostgreSQL об'єднуються з актуальними цінами з оперативної пам'яті сервера. Таким чином, розробка функціональних методів платформи об'єднує складну доменну логіку трейдингу, строгу математику, транзакційне збереження даних та фоновий моніторинг ринку. Модульна структура та чітке розділення відповідальностей суттєво спрощують процес тестування коду за допомогою підставних об'єктів (mock-реалізацій) та забезпечують високу масштабованість системи в майбутньому.

Окрім торгового ядра, невід'ємною частиною розробки є алгоритми управління користувачами, контролю доступу та забезпечення безпеки платформи, які реалізовані в сервісних класах UsersService та ProfileService. Ці модулі відповідають за життєвий цикл профілю трейдера від моменту реєстрації до налаштування параметрів захисту.

Фундаментальним етапом є алгоритм реєстрації та ініціалізації нового трейдера. Процес починається з суворої валідації вхідних об'єктів передачі даних (DTO) та обов'язкової перевірки унікальності електронної пошти. Для виконання цього завдання було розроблено окремий ізольований метод checkEmail, який взаємодіє з базою даних через Prisma ORM.

Лістинг 2.1 – Алгоритм валідації унікальності електронної пошти.

```
private async checkEmail(email: string): Promise<void> {  
  const user = await this.prisma.user.findFirst({  
    where: { email: { equals: email, mode: 'insensitive' } },  
    select: { id: true },  
  });  
  
  if (user) {  
    throw new BadRequestException('User with the same email already exists');  
  }  
}
```

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				38
Змн.	Арк.	№ докум.	Підпис	Дата		

Як видно з наведеного лістингу, метод використовує параметр `mode: 'insensitive'`. Це архітектурне рішення є критично важливим для безпеки, оскільки воно виконує реєстронезалежний пошук і надійно запобігає колізіям (наприклад, система сприйматиме `Test@mail.com` та `test@mail.com` як єдиний ідентифікатор). Крім того, з метою оптимізації продуктивності та зменшення навантаження на базу даних, застосовується конструкція `select: { id: true }`. Замість того, щоб завантажувати в оперативну пам'ять весь об'єкт профілю з усіма зв'язками, сервер отримує лише ідентифікатор, що суттєво пришвидшує роботу методу при великій кількості одночасних реєстрацій.

Після успішної валідації пошти ініціюється процес безпосереднього створення акаунта.

Лістинг 2.2 — Алгоритм реєстрації користувача.

```
async create(data: CreateUserDTO): Promise<ViewUserDTO> {  
  await this.checkEmail(data.email);  
  const newUserId = randomUUID();  
  const user = await this.prisma.user.create({  
    data: { id: newUserId, email: data.email, status: UserStatus.active },  
  });  
  const reset = await this.passwordResetService.createOrReplace(user.id);  
  await this.helloEmailService.send({ ...reset, email: user.email });  
  return this.mapper.mapOne(user);  
}
```

В основі цього методу лежить генерація криптографічно стійкого унікального ідентифікатора (UUID) замість класичних цілочисельних значень з автоінкрементом. Такий підхід унеможливорює атаки типу "перебір ідентифікаторів" (ID enumeration) і не дозволяє зловмисникам дізнатися реальну кількість зареєстрованих користувачів платформи.

Відмінною рисою цього алгоритму є асинхронний ланцюжок подальших дій, який реалізує безпечний підхід «Passwordless Registration Flow». Замість

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		39

передачі відкритого пароля через мережу під час реєстрації, система одразу генерує безпечний токен доступу (createOrReplace) та ініціює відправку електронного листа з посиланням для встановлення первинного пароля. Це змушує користувача підтвердити факт володіння поштовою скринькою ще до того, як він зможе авторизуватися на платформі, що радикально знижує кількість спам-акаунтів.

Для управління життєвим циклом безпеки також розроблено алгоритм відновлення доступу. Головною його особливістю є механізм багаторівневої перевірки системних статусів та захист від розвідки.

Лістинг 2.3 — Алгоритм безпечного відновлення доступу.

```
public async resetPassword(email: string): Promise<void> {  
    const user = await this.prisma.user.findFirst({  
        where: { email: { equals: email, mode: 'insensitive' } },  
    });  
    if (!user) return;  
    if (user.status !== UserStatus.active) {  
        throw new ForbiddenException('Account is banned');  
    }  
    const reset = await this.passwordResetService.createOrReplace(user.id);  
    await this.helloEmailService.send({ ...reset, email: user.email });  
}
```

Першим рівнем захисту в цьому методі є запобігання витоку інформації про існування акаунта (Account Enumeration Prevention). Якщо введена адреса не знайдена в базі даних, алгоритм тихо перериває виконання операції (if (!user) return;), не повертаючи жодної помилки на клієнт. Завдяки цьому злоумисник не може використовувати форму відновлення пароля для перевірки того, чи зареєстрована конкретна людина на платформі.

Другим рівнем є перевірка стану облікового запису. Якщо профіль було заблоковано адміністратором за порушення правил торгівлі або підозрілу

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				40
Змн.	Арк.	№ докум.	Підпис	Дата		

активність (статус не дорівнює active), алгоритм миттєво генерує виняток ForbiddenException. Це гарантує, що заблоковані користувачі не зможуть обійти бан шляхом генерації нових токенів відновлення доступу. Тільки після проходження всіх цих перевірок система генерує криптографічний токен та передає завдання поштовому мікросервісу.

Після реалізації механізмів безпеки та управління обліковими записами, критично важливим етапом стала побудова аналітичного ядра системи — модуля прогнозування ринкових трендів на основі ансамблів нейромереж. Враховуючи високу волатильність криптовалют, було прийнято рішення реалізувати архітектуру, що поєднує декілька спеціалізованих LSTM-моделей, які тренуються на різних часових інтервалах: 15 хвилин, 1 година та 1 доба.

Центральним функціональним вузлом цього модуля є метод predictEnsemble. Його архітектура побудована за принципом «зваженого голосування», що дозволяє мінімізувати вплив випадкового ринкового шуму. Метод паралельно опитує базові моделі, мінімізуючи затримку обробки запиту користувача, після чого агрегує їхні результати згідно з ваговими коефіцієнтами.

Лістинг 2.4 — Обчислення зваженого тренду ансамблю.

```
const ensembleScore =  
    getScore(pred15m.trend) * weights['15m'] +  
    getScore(pred1h.trend) * weights['1h'] +  
    getScore(pred1d.trend) * weights['1d'];
```

Для забезпечення високої точності кожна базова модель (LSTM) пройшла етап навчання на 1000-денному історичному датасеті BTC-USD. Архітектура мережі включає два шари LSTM (по 50 юнітів) з інтегрованим Dropout-регулюванням для запобігання перенавчанню.

Лістинг 2.5 — Створення та налаштування моделі прогнозування.

```
model = Sequential()  
model.add(LSTM(units=50,                                return_sequences=True,  
input_shape=(x_train.shape[1], 1)))
```

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				41
Змн.	Арк.	№ докум.	Підпис	Дата		

```

model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

```

Процес навчання включав підготовку даних за допомогою MinMaxScaler, що нормалізує ціни в діапазон [0, 1], та нарізання даних на часові вікна (sequence_length = 60). Оптимізація проводилася за допомогою алгоритму Adam, що дозволило моделі швидко конвергувати та ефективно розпізнавати патерни руху ціни.

Для підвищення надійності прогнозів у метод predictEnsemble вбудовано алгоритм визначення консенсусу: якщо всі моделі демонструють однаковий вектор руху, система підвищує показник впевненості (Confidence) на 10%. Фінальний результат, що включає очікуваний тренд, ступінь впевненості та прогнозовану ціну, передається на клієнтський інтерфейс. Така інфраструктура дозволяє легко масштабувати аналітичний модуль, додаючи нові джерела даних або типи моделей без необхідності перебудови торгового ядра системи.

Висновки до другого розділу

У другому розділі було детально розглянуто процеси проєктування бази даних, розробки загальної архітектури веб-додатку та імплементації ключових функціональних методів для платформи симуляції криптовалютних торгів «Trading Simulator Platform». Під час реалізації системи було застосовано передові підходи до розробки програмного забезпечення, зокрема принципи модульної архітектури, патерни впровадження залежностей (Dependency Injection) та ізоляції обчислювальних процесів. Особливий акцент було зроблено на забезпеченні транзакційної цілісності фінансових операцій та створенні відмовостійкої інфраструктури, здатної обробляти інтенсивні потоки даних у режимі реального часу.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				42
Змн.	Арк.	№ докум.	Підпис	Дата		

Проектування бази даних на основі PostgreSQL дозволило створити надійну реляційну структуру, що складається з ключових взаємопов'язаних сутностей, таких як профілі користувачів, віртуальні гаманці, торгові позиції та історія транзакцій. Найважливішим рішенням на цьому етапі стало використання спеціалізованого типу даних Decimal для всіх без винятку фінансових показників. Це усунуло будь-які ризики виникнення алгоритмічних помилок, пов'язаних із числами з плаваючою комою, що є критично важливою вимогою для побудови надійних торгових та банківських платформ. Для забезпечення стабільності та портативності додатку було успішно впроваджено систему контейнеризації Docker. Завдяки використанню інструменту Docker Compose вдалося повністю ізолювати клієнтську частину, серверне ядро, базу даних та веб-сервер Nginx у незалежні контейнери, об'єднані захищеною віртуальною мережею. Такий підхід не лише підвищив загальний рівень кібербезпеки системи, запобігаючи несанкціонованому прямому доступу до бази даних, але й суттєво спростив процеси локального розгортання та заклав міцний фундамент для майбутнього горизонтального масштабування платформи на хмарних серверах.

Головну увагу в програмній реалізації було приділено розробці функціональних методів серверного ядра за допомогою фреймворку NestJS. Центральним елементом бізнес-логіки виступив модуль TradeService, який інкапсулює складні алгоритми обробки торгових ордерів, перевірки балансів та розрахунку нереалізованого прибутку чи збитку (PnL). Інтеграція патерну одиниці роботи (Unit of Work) через механізм ізольованих транзакцій Prisma ORM забезпечила строгу атомарність операцій: списання коштів з віртуального гаманця та створення нової торгової позиції відбуваються виключно синхронно. Це архітектурне рішення повністю виключає можливість порушення консистентності даних або виникнення стану гонитви (race conditions) при одночасному надсиланні кількох торгових запитів користувачем. Додатковий рівень надійності всіх математичних обчислень на рівні серверного коду було реалізовано завдяки застосуванню бібліотеки decimal.js.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Важливою архітектурною особливістю розробленого рішення стала глибока інтеграція двостороннього зв'язку через протокол WebSockets (за допомогою socket.io) для отримання та безперервної трансляції ринкових котирувань. Створений мікросервіс PriceService гарантує, що торгове ядро завжди оперує найактуальнішими даними, які миттєво відображаються на клієнтському терміналі. Крім того, для повної автоматизації торгового процесу було розроблено та успішно впроваджено автономний фоновий процес. Цей механізм безперервно функціонує в оперативній пам'яті сервера, щосекунди аналізуючи поточні ціни криптоактивів і автоматично ініціюючи закриття відкритих позицій при досягненні встановлених користувачем захисних тригерів Stop Loss або цільових рівнів Take Profit. Завдяки цьому платформа працює автономно та гарантує виконання торгових стратегій навіть тоді, коли вкладка браузера у користувача закрита.

Клієнтська частина, побудована на базі сучасного фреймворку Next.js та бібліотеки React з використанням TypeScript, довела свою високу ефективність завдяки реактивній архітектурі, що дозволяє миттєво реагувати на зміни ринку без перезавантаження сторінки. Застосування кастомних React-хуків (наприклад, useChartArrows) для взаємодії з DOM-елементами інтерактивних графіків lightweight-charts, а також використання recharts для візуалізації кругових діаграм портфеля, створили глибоко занурювальний та інтуїтивно зрозумілий інтерфейс трейдера. Стилзація фронтенду за допомогою Tailwind CSS дозволила реалізувати сучасний візуальний досвід із застосуванням ефектів glassmorphism, що відповідає найкращим UI/UX практикам створення фінансових застосунків, знижуючи когнітивне навантаження на користувача.

У підсумку можна з упевненістю стверджувати, що спроектована архітектура, реляційна структура бази даних та реалізовані функціональні методи «Trading Simulator Platform» повністю відповідають найсуворішим індустріальним стандартам розробки складних веб-додатків. Обраний високотехнологічний стек забезпечує чітке розділення відповідальностей між

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				44
Змн.	Арк.	№ докум.	Підпис	Дата		

клієнтом та сервером, високу розширюваність завдяки модульній структурі NestJS та максимальний рівень безпеки завдяки JWT-автентифікації і транзакційному виконанню запитів. Це підтверджує абсолютну готовність розробленого програмного рішення до інтенсивного використання як навчальної та аналітичної платформи, яка здатна стабільно витримувати високі навантаження та надавати користувачам бездоганний і безпечний досвід симуляції реальних ринкових торгів.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				45
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК ВИКОРИСТАННЯ ДОДАТКУ

3.1 Опис роботи з програмним додатком (Опис інтерфейсу)

На зображеннях представлено інтерфейси сторінок реєстрації, авторизації, головного торгового терміналу та особистого портфеля веб-платформи «Trading Simulator Platform». Нижче наведено докладний опис та аналіз реалізації кожної з цих сторінок у контексті розробки повноцінної системи симуляції торгівлі криптоактивами, яка є практичним результатом виконання дипломного проєкту.

Після переходу на сайт користувач потрапляє на сторінку авторизації, де може увійти до системи, використовуючи свої облікові дані. Для зручності та безпеки реалізовано функцію відновлення доступу, а також передбачено безпечний механізм реєстрації нових трейдерів. Сторінка авторизації має сучасний та лаконічний інтерфейс, виконаний у темній темі з використанням візуальних ефектів glassmorphism (напівпрозорі елементи з розмиттям фону), що підтримує єдиний преміальний стиль фінансової платформи. Інтерфейс сторінки авторизації виглядає таким чином (рис. 3.1).

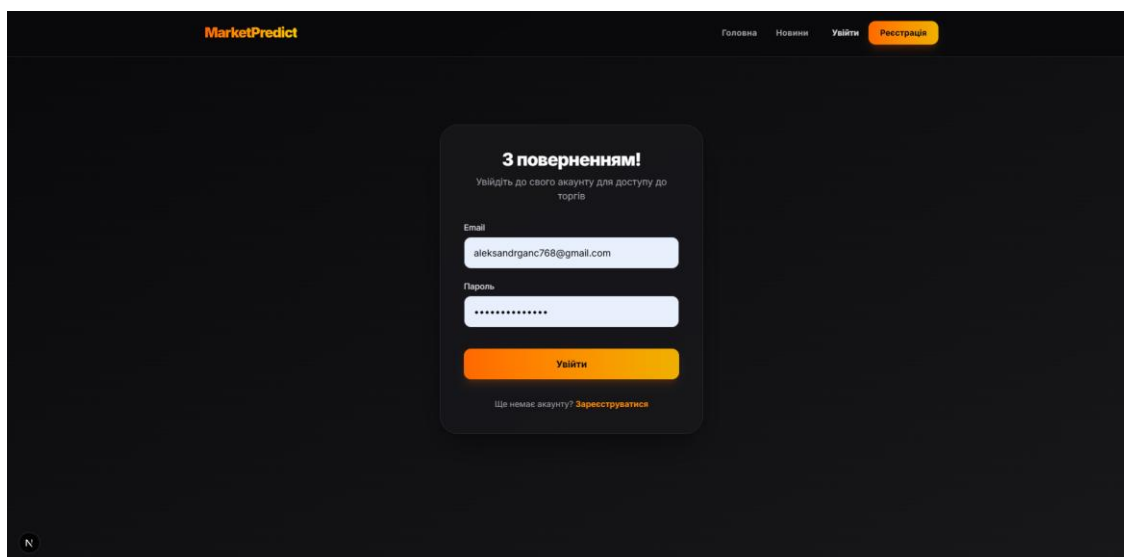


Рисунок 3.1 — Сторінка авторизації

На зображенні представлено сторінку входу до системи, де користувач вводить свою електронну пошту та пароль. Форма входу містить чітко виділені поля для вводу, контрастну кнопку підтвердження та посилання для скидання

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				46
Змн.	Арк.	№ докум.	Підпис	Дата		

пароля у разі його втрати. Дизайн сторінки орієнтований на мінімізацію відволікаючих факторів, що дозволяє користувачеві швидко сфокусуватися на вході. Після успішної авторизації та перевірки tokenів безпеки користувач автоматично перенаправляється до свого віртуального робочого простору.

Для нових користувачів передбачено спеціальну процедуру реєстрації, процес якої відрізняється підвищеним рівнем безпеки та верифікації порівняно зі стандартними формами. Інтерфейс першого кроку реєстрації є максимально спрощеним. Сторінка реєстрації виглядає наступним чином (рис. 3.2).

Рисунок 3.2 — Сторінка реєстрації

На зображенні показано початковий етап створення нового облікового запису. Замість традиційного заповнення об'ємної форми з введенням паролів безпосередньо на сайті, користувач вводить лише свій email та натискає кнопку підтвердження. Після цього система бекенду генерує унікальний захищений токен і автоматично надсилає на вказану адресу електронний лист із посиланням. Перейшовши за цим посиланням, користувач потрапляє на окрему захищену сторінку завершення реєстрації, де він вже самостійно створює та підтверджує свій пароль. Такий архітектурний підхід не лише гарантує, що вказана електронна адреса дійсно належить живій людині, але й ефективно захищає платформу від масового створення фейкових акаунтів автоматизованими скриптами. Важливою особливістю інтерфейсу також є наявність зручного посилання для швидкого

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				47
Змн.	Арк.	№ докум.	Підпис	Дата		

переходу на сторінку логіну для вже існуючих користувачів. Усі форми розміщені по центру екрана, що візуально виокремлює їх з темного фону й створює враження чистоти та технологічності інтерфейсу.

Після успішної автентифікації головним робочим простором користувача стає сторінка торгового терміналу (Market Page). Ця сторінка виконує роль високотехнологічного центру для аналізу ринку та здійснення віртуальних торгових операцій у режимі реального часу. Основну частину екрана займає інтерактивний графік котирувань, а поруч розташована функціональна панель управління ордерами. Інтерфейс торгового терміналу виглядає таким чином (рис. 3.3).

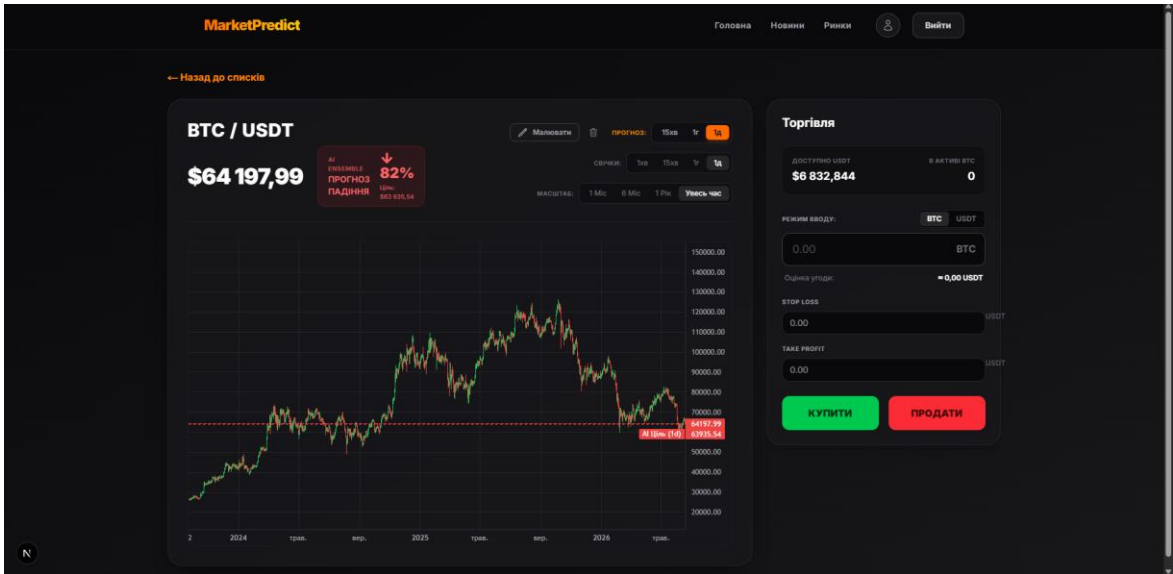


Рисунок 3.3 — Сторінка торгового терміналу

На зображенні представлено сторінку торгового терміналу, яка дозволяє користувачеві детально аналізувати обраний криптоактив. Центральним елементом є графік японських свічок, реалізований за допомогою бібліотеки `lightweight-charts`. Він безперервно оновлюється завдяки інтеграції `WebSockets`. Користувач має можливість перемикати таймфрейми (наприклад, 1 хвилина, 15 хвилин, 1 година, 1 день) для аналізу як короткострокових, так і довгострокових трендів ринку. Поруч із графіком розміщено панель купівлі та продажу (Buy/Sell), де трейдер може вказати обсяг угоди, встановити автоматичні захисні ордери `Stop Loss` та `Take Profit`. Додатковою особливістю цього інтерфейсу є візуальне відображення роботи алгоритмів предиктивної аналітики, яка безпосередньо на

графіку показує ймовірність зміни тренду та орієнтовну цільову ціну за допомогою кастомних маркерів. Завдяки такому професійному, але не перевантаженому інтерфейсу, користувачі мають змогу швидко аналізувати ситуацію та ефективно ухвалювати торгові рішення.

Для відстеження власної фінансової ефективності та управління відкритими позиціями розроблено сторінку особистого кабінету та портфеля (Portfolio Page). Ця сторінка є ключовим аналітичним вузлом для трейдера, оскільки вона агрегує всю інформацію про його віртуальний капітал і поточні успіхи. Сторінка управління портфелем виглядає наступним чином (рис. 3.4).

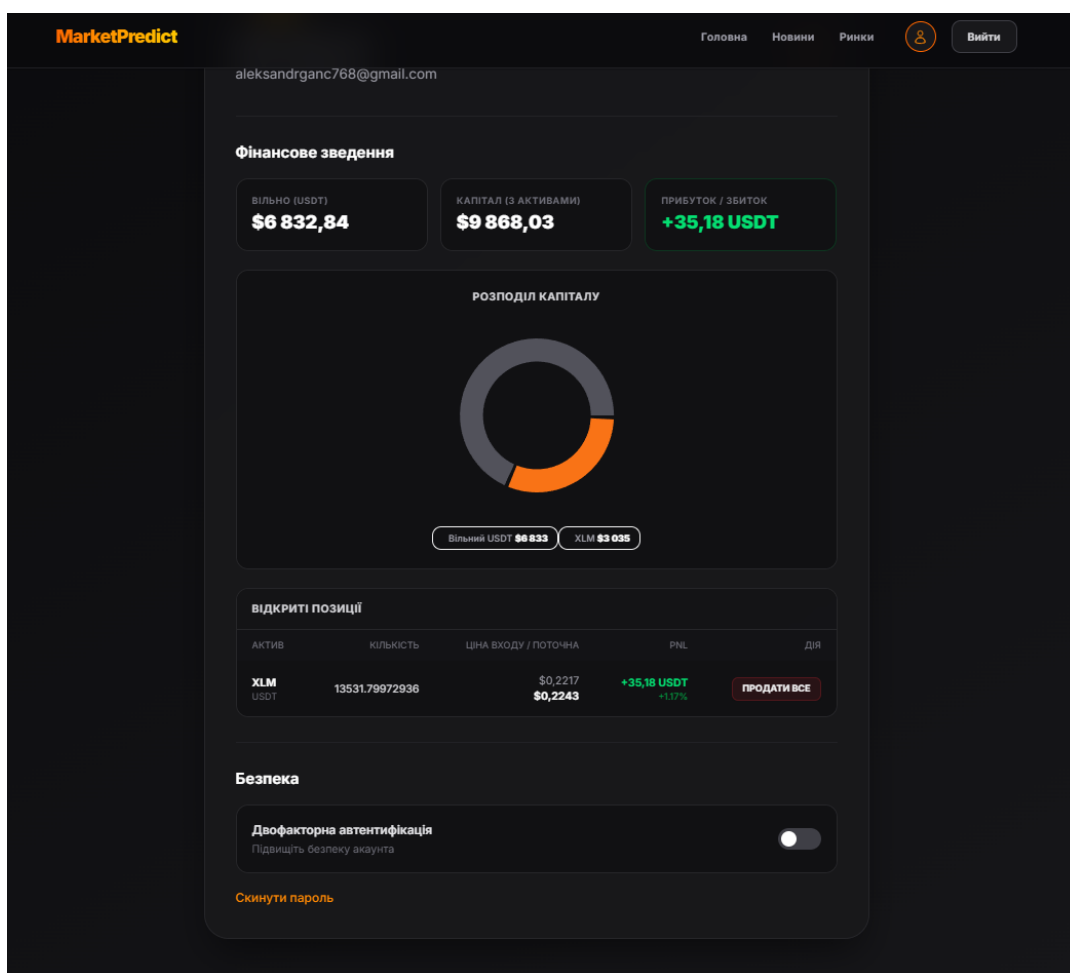


Рисунок 3.4 — Сторінка управління портфелем

На зображенні показано сторінку особистого кабінету. У верхній частині інтерфейсу відображається загальна оцінка капіталу користувача в еквіваленті USDT, яка динамічно перераховується щосекунди залежно від коливань ринкових цін на активи у відкритих позиціях. Центральне місце займає інтерактивна

кругова діаграма, побудована за допомогою бібліотеки *recharts*, яка наочно демонструє відсотковий розподіл коштів між різними криптовалютами в портфелі. При наведенні курсору на сектори діаграми з'являються кастомні підказки з детальною інформацією. Нижче розташована структурована таблиця активних торгових позицій, де для кожної угоди автоматично розраховується абсолютний та відсотковий нереалізований прибуток або збиток (PnL). Для оперативного реагування на екстремальні зміни ринку в інтерфейсі таблиці передбачено кнопку швидкого закриття активних угод. Загалом, структура цієї сторінки розроблена з урахуванням принципів UX-дизайну, що дозволяє трейдеру миттєво оцінити стан свого рахунку без необхідності переходити між різними вкладками.

3.2 Роутинг та маршрутизація

Для забезпечення структурованої та безпечної взаємодії між клієнтським інтерфейсом та серверною частиною системи було розроблено архітектуру програмного інтерфейсу (API), що базується на принципах REST (Representational State Transfer) [18]. Наведена нижче специфікація демонструє детальну логіку роутингу та маршрутизації HTTP-запитів, які обслуговують ключову бізнес-логіку платформи «Trading Simulator Platform» — обробку ринкових даних та здійснення фінансових операцій.

Представлена структура описує чіткий розподіл відповідальностей між серверними контролерами. Зокрема, вона деталізує механізми обробки інформаційних GET-запитів: отримання глобального списку відстежуваних криптовалютних пар, завантаження агрегованих історичних даних для високоточної побудови графіків японських свічок на клієнті, а також маршрутизацію запитів до модуля предиктивної аналітики для отримання миттєвих прогнозів штучного інтелекту. Окремий логічний блок присвячений транзакційним POST та PATCH запитам, які безпосередньо ініціюють створення торгових ордерів (купівля чи продаж активів), встановлення лімітів Stop Loss та

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				50
Змн.	Арк.	№ докум.	Підпис	Дата		

Take Profit, а також агрегацію даних для динамічного управління портфелем користувача.

Важливою архітектурною складовою цієї маршрутизації є застосування проміжних шарів валідації та безпеки (Guards). Абсолютна більшість ендпоінтів (маршрутів), пов'язаних із торговим ядром, є захищеними і вимагають обов'язкової валідації JWT-токена в заголовках запиту, що гарантує сувору авторизацію кожної фінансової дії. Загалом, ця таблиця є критично важливою для комплексного розуміння того, як саме налаштовано дуплексну взаємодію між клієнтським додатком і бекендом за допомогою контролерів та об'єктів передачі даних (DTO) під час роботи з високоволатильним фінансовим контентом у реальному часі (табл. 3.1).

Таблиця 3.1

Специфікація API-маршрутів та їх відображення в структурі вебзастосунку

HTTP Route	Next.js Route	Дія
GET /market/symbols	/market	Перелік відстежуваних криптовалют
GET /market/history/{symbol}	/market/[symbol]	Отримання історичних даних для графіка
GET /market/predict/{symbol}	/market/[symbol]	Отримання миттєвого II-прогнозу для монети
POST /trade/order	/market/[symbol]	Створення торгового ордера (Buy/Sell/SL/TP)

продовж. табл. 1.2

GET /trade/portfolio	/profile	Отримання даних портфеля та розрахунків PnL
GET /trade/history	/profile	Отримання історії торгових угод
GET /market/stats/24h	/	Сводка за 24 години для швидкого завантаження дашборда

Перед вами таблиця, що містить роутинг та маршрутизацію для запитів, що стосуються авторизації, управління профілями та адміністрування користувачів на платформі. Вона описує маршрути для логіну, безпечного підтвердження реєстрації, отримання та оновлення даних поточного профілю, налаштування двофакторної автентифікації (2FA), а також адміністративні маршрути для перегляду списку користувачів, їх блокування (ban) або нарахування віртуального депозиту. Ці маршрути є фундаментальними для забезпечення безпечного доступу до системи, розмежування прав між звичайними трейдерами та адміністраторами, а також для повноцінного управління обліковими записами. Реалізований механізм верифікації та налаштування безпеки гарантує захист фінансової статистики кожного учасника платформи (табл. 3.2).

Таблиця 3.2

Специфікація API-маршрутів авторизації, управління профілями та адміністрування користувачів

HTTP Route	Next.js Route	Дія
POST /auth/login	/login	Авторизація користувача (вхід)

POST /auth/verify	/verify	Підтвердження реєстрації та встановлення пароля
GET /profile	/profile	Отримання даних поточного користувача
PATCH /profile/2fa	/profile	Налаштування двофакторної автентифікації
POST /profile/reset	/reset-password	Запит на скидання пароля
GET /users	/admin/users	Отримання переліку всіх користувачів (для адміна)
POST /users/{id}/ban	/admin/users	Блокування профілю користувача
POST /trade/deposit/{id}	/admin/users	Нарахування віртуального депозиту користувачу

У проєкті роутинг на серверній частині реалізовано з суворим дотриманням принципів REST-архітектури та структуровано за логічними доменами (ресурсами) за допомогою контролерів фреймворку NestJS. Для кожного ресурсу (наприклад, market, trade, profile, users) визначено чіткий набір маршрутів, що відповідають семантиці основних HTTP-методів: GET використовується виключно для отримання даних без зміни стану системи, POST — для створення нових сутностей (наприклад, ордерів або нових криптовалют для відстеження), PUT/PATCH — для оновлення профілів або налаштувань, а DELETE — для

видалення (наприклад, зняття монети з моніторингу). Завдяки модульній структурі NestJS, така організація API забезпечує високу прозорість бізнес-логіки, полегшує підключення захисних механізмів (Guard) для перевірки JWT-токенів на конкретних ендпоінтах та значно спрощує подальше розширення функціоналу бекенду.

На клієнтській частині маршрутизацію побудовано з використанням сучасної парадигми App Router фреймворку Next.js [7, 29]. На відміну від класичного підходу React Router, тут відтворено деревоподібну структуру на основі файлової системи, що ідеально синхронізується з REST-ресурсами серверного API. Компоненти інтерфейсу логічно розділено на окремі сторінки та вкладені макети (layouts). Наприклад, використання динамічних сегментів у назвах папок, таких як `/market/[symbol]`, дозволяє системі автоматично перехоплювати параметр криптоактиву з URL-адреси та динамічно підвантажувати відповідний графік та панель торгівлі за допомогою запитів до `GET /market/history/{symbol}` та `POST /trade/order`.

Уніфікована та синхронізована структура маршрутів між бекендом та фронтендом сприяє високій масштабованості архітектури. Вона не лише полегшує розробку нових модулів — наприклад, додавання розділу новин (`/news`), де адміністратори створюють анонси через `POST /news/announcements`, а користувачі переглядають їх на відповідній сторінці — але й підвищує загальну зрозумілість коду. Це дозволяє забезпечити швидку навігацію в режимі Single Page Application (SPA), оптимізувати процес передачі даних та створити максимально плавний і безперервний користувацький досвід у торговому терміналі.

3.3 Робота з авторизацією (JWT)

У проєкті симулятора криптовалютних торгів «Trading Simulator Platform» для автентифікації та авторизації користувачів використовується сучасний, надійний та оптимізований механізм на основі JSON Web Token (JWT) [30]. Процес авторизації суворо адаптовано до підвищених вимог безпеки фінансових

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		54

платформ. Після успішного проходження процедури входу (ендпоінт /auth/login) або верифікації пошти та встановлення пароля, сервер перевіряє облікові дані та формує криптографічно підписаний токен. Цей токен містить закодовану інформацію про користувача, зокрема його унікальний ідентифікатор (UUID), електронну адресу, статус двофакторної автентифікації (2FA) та роль у системі (наприклад, адміністратор чи звичайний трейдер). Згенерований JWT передається клієнту для подальшого використання у захищених маршрутах, що дозволяє серверу ідентифікувати користувача без необхідності зберігати інформацію про сесію в оперативній пам'яті, забезпечуючи архітектурну безстейтовість (stateless) взаємодії.

Для генерації JWT на серверній частині застосовується спеціалізований модуль фреймворку NestJS (@nestjs/jwt). Цей процес включає створення корисного навантаження (payload) — об'єкта, що містить ключові ідентифікаційні дані користувача та системні мітки часу, такі як дата створення і точний строк дії токена. Цей об'єкт серіалізується у формат JSON, після чого підписується за допомогою надійного криптографічного алгоритму (зазвичай HMAC SHA256) [10] із використанням секретного ключа, який зберігається у захищених змінних середовища сервера. Структура згенерованого токена складається з трьох послідовних частин, розділених крапкою: заголовка (header), що вказує на тип та алгоритм, корисного навантаження (payload) та криптографічного підпису (signature). Наявність цифрового підпису гарантує, що дані не були змінені, скомпрометовані або підроблені сторонніми особами під час передачі відкритою мережею.

На стороні клієнтського додатку, побудованого на базі React та Next.js, після успішного отримання токена він надійно зберігається для організації безперервної сесії користувача. Залежно від конфігурації, токен зберігається у локальному сховищі браузера (localStorage) або, для забезпечення ще вищого рівня захисту від XSS-атак, у захищених HTTP-only файлах cookie [11]. При зверненні до захищених ресурсів платформи, таких як перегляд історії транзакцій,

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		55

аналіз особистого портфеля або ініціація торгового ордеру на купівлю активу, клієнт автоматично додає цей токен у HTTP-заголовок запиту за схемою `Authorization: Bearer {token}`. У клієнтському коді це ефективно реалізується через глобальне налаштування екземпляра HTTP-клієнта за допомогою перехоплювачів (наприклад, конфігурацією `axios.defaults.headers.common['Authorization']`), що гарантує автоматичне прикріплення ключа доступу до кожного вихідного мережевого запиту.

На сервері реалізовано надійний механізм перехоплення та валідації запитів за допомогою концепції Guards (охоронців маршрутів) у NestJS, зокрема спеціалізованого класу `JwtAuthGuard`. Цей проміжний архітектурний шар перехоплює всі вхідні HTTP-запити до захищених маршрутів (наприклад, `/trade/order` або `/profile`), витягує токен із заголовка та перевіряє його математичну валідність. Серверна перевірка включає криптографічну звірку підпису, контроль терміну дії токена та, за необхідності, перевірку статусу користувача в базі даних PostgreSQL. Якщо токен є дійсним, розкодовані дані корисного навантаження автоматично прикріплюються до об'єкта поточного запиту (зазвичай у `request.user`), після чого управління безпечно передається відповідному контролеру для виконання бізнес-логіки. Якщо токен відсутній, пошкоджений або його термін дії закінчився, система миттєво відхиляє запит, повертаючи клієнту стандартний HTTP-статус помилки 401 Unauthorized або 403 Forbidden.

Окремою критично важливою особливістю використання JWT у розроблюваній торговій платформі є його безшовна інтеграція з протоколом WebSockets. Оскільки торговий термінал вимагає безперервного двостороннього потоку котирувань та миттєвої реакції на зміну балансу користувача, WebSocket-з'єднання (реалізовані через бібліотеку `socket.io`) також проходять сувору процедуру автентифікації. Токен авторизації передається клієнтом під час початкового "рукостискання" (handshake) веб-сокета. Це дозволяє серверному ядру ідентифікувати підключеного клієнта і безпечно транслявати йому персоналізовані події у реальному часі, такі як сповіщення про автоматичне

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		56

спрацювання ордерів Stop Loss або Take Profit, без необхідності встановлення нових HTTP-з'єднань чи повторної перевірки паролів

Загалом, застосування JSON Web Token у поєднанні з модульною архітектурою NestJS забезпечує платформі надійний, високопродуктивний та легко масштабований підхід до керування доступом. Відсутність потреби у централізованому сховищі сесій на сервері значно зменшує навантаження на базу даних та дозволяє гнучко масштабувати бекенд-інфраструктуру при стрімкому зростанні кількості активних трейдерів. Крім того, наявність вбудованих ролей у самому структурованому токени дозволяє елегантно розмежувати права доступу між звичайними користувачами симулятора та адміністраторами, створюючи абсолютно безпечну та керовану модель функціонування сучасного фінансового веб-додатку.

3.4 User Stories та User Flows

У цьому підрозділі розглядаються основні сценарії взаємодії користувача з розробленою платформою «Trading Simulator Platform», а саме процеси безпечної реєстрації, здійснення віртуальних торгових операцій, аналізу ринкових даних та управління особистим інвестиційним портфелем. Це дозволяє краще зрозуміти, як саме користувачі взаємодіють із системою симуляції торгів та якими аналітичними інструментами вони користуються для досягнення своїх цілей у вивченні криптовалютного ринку. Описані потоки роботи надають чітке уявлення про послідовність дій користувачів у різних сценаріях, починаючи від першого знайомства з платформою і закінчуючи налаштуванням складних автоматичних ордерів (Рисунок 3.5).

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				57
Змн.	Арк.	№ докум.	Підпис	Дата		

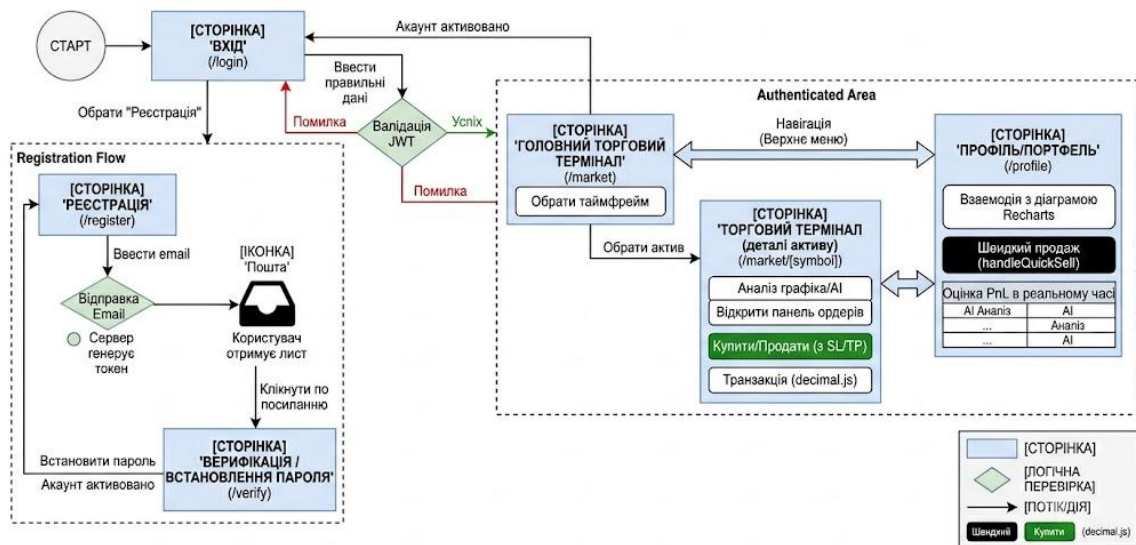


Рисунок 3.5 — Схема потоків користувача

User Stories (користувацькі історії) виступають фундаментальним інструментом проєктування вимог в гнучких методологіях розробки, описуючи основні функції та сценарії взаємодії, які користувачі очікують від платформи для ефективного та безпечного тестування своїх торгових стратегій.

Як неавторизований відвідувач платформи, я хочу мати можливість швидко зареєструватися за допомогою електронної пошти без необхідності одразу вгадувати складні паролі (Passwordless-подібний підхід), щоб максимально швидко створити свій ізольований віртуальний рахунок і розпочати навчання без організаційних перешкод. Як трейдер-початківець, я хочу переглядати високопродуктивні інтерактивні графіки японських свічок у реальному часі та вільно змінювати таймфрейми (від 1 хвилини до 1 дня), щоб якісно оцінювати поточні ринкові тренди методами технічного аналізу. Я також хочу мати можливість відкривати позиції на покупку чи продаж активів, гнучко налаштовуючи захисні рівні Stop Loss (для обмеження потенційних збитків) та Take Profit (для автоматичної фіксації прибутку), щоб серверна система могла автономно закрити угоду без моєї безпосередньої участі, щойно ринкова ціна досягне заданої позначки. Крім того, як користувач, що дбає про безпеку, я хочу мати можливість активувати двофакторну автентифікацію (2FA) у своєму профілі

для додаткового захисту мого віртуального капіталу від несанкціонованого доступу.

Як віртуальний інвестор, я хочу бачити наочний візуальний розподіл мого капіталу на інтерактивній круговій діаграмі та відстежувати нереалізований прибуток або збиток (PnL) у реальному часі для миттєвої оцінки власної фінансової ефективності та диверсифікації портфеля. Як адміністратор платформи, я хочу мати доступ до захищеного інтерфейсу зі списком усіх користувачів, щоб за необхідності блокувати порушників системи або ініціювати транзакції для нарахування додаткових віртуальних депозитів користувачам для продовження їхнього навчання торгівлі. Нарешті, як сучасний користувач FinTech-продуктів, я хочу мати доступ до предиктивної аналітики на базі штучного інтелекту, щоб отримувати візуальні підказки та прогнози щодо ймовірних майбутніх рухів ціни безпосередньо на робочому торговому графіку.

User Flow (потік користувача) описує конкретну логічну послідовність кроків та станів системи, які проходить трейдер для реалізації своїх завдань на платформі.

Процес онбордингу та реєстрації починається з того, що відвідувач заходить на сторінку /register, вводить свою електронну адресу та ініціює створення акаунту. Замість класичного миттєвого введення пароля, серверна система (NestJS) виконує валідацію на відсутність дублікатів, генерує захищений криптографічний токен і автоматично надсилає користувачеві транзакційний лист. Перейшовши за унікальним одноразовим посиланням з листа, користувач потрапляє на маршрут клієнтського додатку /set-password, де самостійно встановлює та підтверджує свій пароль. На цьому етапі бекенд хешує пароль і остаточно активує обліковий запис. Після цього трейдер переходить на сторінку /login, вводить свої облікові дані та, у разі успішної перевірки, отримує від сервера авторизаційні JWT-токени (Access та Refresh). Клієнтський додаток (Next.js) зберігає сесію та автоматично перенаправляє користувача до головного торгового терміналу на маршрут /market.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				59
Змн.	Арк.	№ докум.	Підпис	Дата		

Для здійснення торгових операцій користувач обирає потрібну криптовалютну пару і переходить на її динамічну сторінку (наприклад, /market/BTC). На цьому етапі клієнтський додаток ініціює дуплексне WebSocket-з'єднання, що дозволяє безперервно оновлювати графік котирувань без перезавантаження сторінки. Трейдер аналізує ринкову ситуацію та ознайомлюється з AI-прогнозами, інтегрованими в інтерфейс. Переконавшись у своєму рішенні, він взаємодіє з панеллю ордерів: вводить бажаний обсяг інвестиції (система миттєво валідує достатність балансу), за потреби налаштовує ліміти для автоматичного закриття позиції та натискає кнопку підтвердження угоди (Buy або Sell). У разі успішного виконання транзакції користувач отримує візуальне сповіщення (Toast Notification) про відкриття позиції.

Для глобального управління вже відкритими позиціями та аудиту свого загального прогресу користувач переходить до розділу особистого профілю (/profile). Тут він взаємодіє з інтерактивною круговою діаграмою розподілу активів, наводячи курсор на окремі сектори для перегляду точних сум у спливаючих підказках. Нижче трейдер аналізує таблицю активних ордерів, де показник PnL (Profit and Loss) динамічно перераховується системою на основі поточних ринкових цін. У випадку різкої непередбачуваної зміни ситуації на ринку (Flash Crash), трейдер може використати функцію швидкого продажу (Quick Sell), натиснувши відповідну кнопку для пакетного закриття всіх активних позицій по обраному активу за поточною ринковою ціною, фіксує результат у єдиній базі даних транзакцій.

3.5 Заходи безпеки бази даних

Під час роботи з реляційною системою управління базами даних PostgreSQL було реалізовано комплекс багаторівневих заходів безпеки, які сприяють захисту конфіденційної інформації та кардинальному зниженню ризику несанкціонованого доступу. Серед основних механізмів захисту на рівні архітектури застосовано надійне криптографічне хешування паролів (із використанням сучасних алгоритмів генерації ключів, стійких до атак на базі

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				60
Змн.	Арк.	№ докум.	Підпис	Дата		

графічних процесорів) перед їх збереженням у базі. Додатково впроваджено використання криптографічно стійкого формату UUID (версії 4) як унікального ідентифікатора (Primary Key) для всіх ключових сутностей. Це архітектурне рішення повністю унеможливорює атаки типу перебору записів (ID enumeration) та приховує від потенційних зломисників загальні обсяги реєстрацій чи угод на платформі. На рівні самої СКБД також реалізовано суворе налаштування ролей та привілеїв, що базується на принципі найменших привілеїв (Principle of Least Privilege), гарантуючи безпечне розмежування доступу до таблиць. Окрім цього, для захисту даних у процесі передачі між додатком і базою даних передбачено використання зашифрованих з'єднань (SSL/TLS), що виключає можливість перехоплення мережевого трафіку (Man-in-the-Middle).

База даних PostgreSQL обґрунтовано вважається індустріальним стандартом для фінансових додатків [18], оскільки вона вирізняється безпрецедентною надійністю та масштабованістю. Система повністю підтримує ACID-транзакції (атомарність, узгодженість, ізолюваність, довговічність) та механізм багатоверсійного управління конкурентним доступом (MVCC), що дозволяє обробляти тисячі паралельних торгових запитів [22] без блокування таблиць для читання. Крім того, PostgreSQL підтримує ефективне зберігання напівструктурованих даних у форматі JSONB та використання складних тригерів, що робить її ідеальною для реалізації нетипових сценаріїв фінансової платформи.

Для абстракції бази даних, управління її структурою та виконання запитів замість традиційних SQL-рішень використовується сучасна Prisma ORM. Цей інструмент гарантує абсолютну безпеку взаємодії з даними завдяки автоматичній генерації строго типізованого клієнта на етапі компіляції. Такий підхід на архітектурному рівні нівелює ризики SQL-ін'єкцій, оскільки всі змінні автоматично параметризуються. Критично важливою функцією Prisma для симулятора торгів є підтримка інтерактивних транзакцій (Interactive Transactions), які гарантують атомарне виконання серії операцій — наприклад, одночасне списання коштів з балансу та створення торгового ордеру. У разі виникнення

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				61
Змн.	Арк.	№ докум.	Підпис	Дата		

помилки на будь-якому етапі транзакції всі зміни автоматично відкочуються (Rollback), що унеможливорює розсинхронізацію фінансових даних. Керування схемою здійснюється через детермінований механізм міграцій, а для зручного візуального перегляду, адміністрування та аналітики розгорнуто графічний інтерфейс Prisma Studio, який слугує сучасним веб-аналогом класичних десктопних клієнтів.

Усі параметри інфраструктури зберігання даних детально описані та задекларовані в конфігураційному файлі інструменту оркестрації Docker Compose. Контейнер бази даних запускає офіційний сервер PostgreSQL і використовує захищені змінні оточення (зчитувані з файлу .env) для безпечної ініціалізації бази, користувача та пароля без їх хардкодингу у вихідному коді. Щоб унеможливити втрату фінансових даних користувачів та історії угод між перезапусками або оновленнями системи, використовується механізм постійних томів (Docker Volumes), який зберігає фізичні файли БД на хост-машині. Усі контейнери додатку підключені до спільної ізольованої віртуальної мережі (bridge network), у межах якої сервісам призначаються внутрішні DNS-адреси. Це гарантує, що стандартний порт бази даних не виставляється напряму в публічний доступ інтернету, а взаємодіє виключно із серверним API-ядром, що створює закритий і максимально захищений мережевий периметр. Додатково в конфігурації Docker Compose налаштовано механізм перевірки працездатності (Healthchecks), який гарантує, що бекенд-сервіс на NestJS почне ініціалізацію підключення лише після повного запуску та готовності бази даних приймати з'єднання.

Використання технології Docker у процесі розробки та продуктивного розгортання програмних рішень на базі середовища Node.js із використанням фреймворку NestJS забезпечує низку суттєвих інженерних переваг. Головною з них є повна декларативна автоматизація інфраструктури: завдяки конфігурації docker-compose.yml можна розгорнути повноцінне робоче середовище (включаючи API, PostgreSQL та Prisma Studio) однією термінальною командою

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		62

без складного ручного налаштування кожного компонента. Такий підхід ідеально інтегрується в сучасні CI/CD-процеси та забезпечує швидке розгортання на хмарних серверах. Додатково використання оптимізованих образів гарантує сувору уніфікацію середовищ для розробки та продакшену, повністю усуваючи класичну проблему несумісності між різними операційними системами або локальними версіями Node.js, що радикально зменшує витрати часу на відлагодження проєкту.

Особливу увагу при розгортанні платформи приділено архітектурі побудови самих Docker-образів за допомогою багатоетапних збірок (multi-stage builds) [20]. Цей підхід дозволяє використовувати важкі базові образи з усіма необхідними інструментами компіляції TypeScript на етапі збирання проєкту, а у фінальний продуктивний образ копіювати лише скомпільований JavaScript-код та необхідні для виконання залежності. Це не лише суттєво зменшує кінцевий розмір контейнера, але й мінімізує поверхню для потенційних кібератак, виключаючи інструменти розробника з робочого середовища.

Крім того, контейнеризація забезпечує надійну ізоляцію сервісів та жорсткий контроль за споживанням апаратних ресурсів (лімітування CPU та оперативної пам'яті). Кожен логічний компонент платформи працює у власному незалежному контейнері, що дозволяє швидко ідентифікувати джерело потенційних проблем і гарантує, що помилка (наприклад, витік пам'яті) в одному модулі не призведе до каскадного падіння інших частин системи. Docker також надає гнучкі можливості для стандартизації логування та тестування, дозволяючи легко запускати інтеграційні тести в ефемерному (тимчасовому) середовищі, яке на 100% відтворює продуктивну інфраструктуру. У підсумку, використання Docker у проєктах на базі сучасного стеку NestJS та Prisma ORM трансформує архітектуру в Cloud-Native формат, що значно покращує стабільність, горизонтальну масштабованість і загальну керованість торгової платформи.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				63
Змн.	Арк.	№ докум.	Підпис	Дата		

3.6 Застосування Postman для адміністрування та тестування API

Для забезпечення безперебійної взаємодії між клієнтською та серверною частинами платформи «Trading Simulator Platform», а також для оптимізації процесу відлагодження, було створено комплексну інтерактивну документацію API за допомогою індустріального стандарту — інструменту Postman. Наведений фрагмент документації наочно демонструє структуровану колекцію HTTP-запитів, яка логічно розділена на домени та повністю покриває ключові бізнес-процеси платформи. Окремий акцент зроблено на ендпоінті створення торгового ордера, який є центральним елементом торгового ядра і реалізується через POST-запит. Інтерфейс Postman дозволяє не лише переглядати специфікацію, але й виконувати наскрізне тестування API-ендпоінтів, перевіряючи їхню коректність та час відгуку у реальному часі без необхідності залучення графічного інтерфейсу клієнтського додатка. Документація суворо регламентує структуру тіла запитів (Payload) у форматі JSON, визначає обов'язкові об'єкти передачі даних (DTO) та заголовки безпеки (Headers), зокрема передачу валідних токенів авторизації. Така деталізація перетворює колекцію на надійний технічний контракт між мікросервісами системи. Зображення робочого простору інструменту виглядає наступним чином (рис. 3.6).

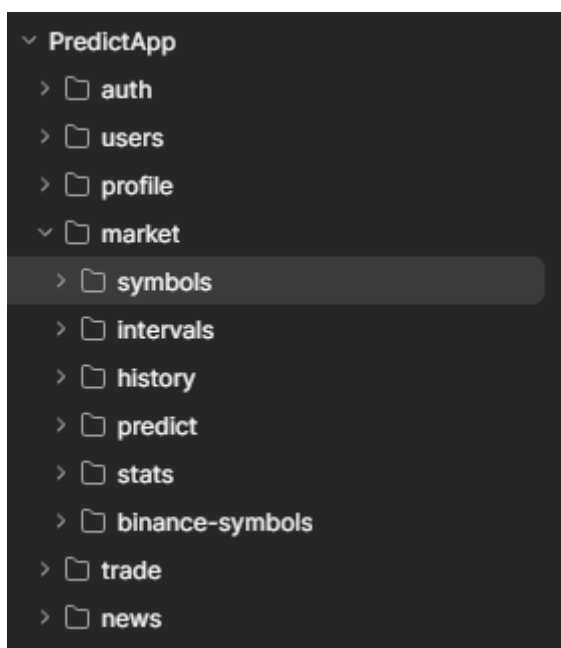


Рисунок 3.6 – Документація роутів Postman для торгового терміналу

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				64
Змн.	Арк.	№ докум.	Підпис	Дата		

У процесі розробки сучасних фінансових вебзастосунків особливу увагу приділяють проєктуванню та підтримці надійного API, яке виступає посередником між клієнтським торговим терміналом та серверним ядром на базі NestJS. Одним із найбільш потужних інструментів для взаємодії з бекендом є Postman — багатофункціональне середовище для тестування та документування RESTful [31] сервісів. У рамках розробки дипломного проєкту Postman активно використовувався як основний засіб для перевірки правильності реалізації маршрутизованих запитів, зокрема при взаємодії з модулями обробки ринкових даних, управління портфелем та виконання торгових операцій. Завдяки інтуїтивному інтерфейсу, Postman дозволяє моделювати запити з різними HTTP-методами (GET, POST, PUT, PATCH, DELETE), встановлювати необхідні заголовки, а також передавати дані у форматі JSON. Усі запити структуруються в колекції, що дозволяє розробнику швидко орієнтуватися у великій кількості ендпоінтів. Крім того, програма підтримує змінні середовища, які суттєво спрощують тестування платформи у різних умовах, таких як локальна розробка чи розгорнутий продуктивний сервер.

Однією з найцінніших функцій для тестування безпеки фінансової системи є можливість автоматизованої авторизації запитів через JWT-токени. Це дозволяє перевірити, як система реагує на дії автентифікованих трейдерів та спроби несанкціонованого доступу, включно з можливістю створення ордерів, перегляду історії транзакцій або налаштування двофакторної автентифікації. Для прикладу, при тестуванні запиту POST /trade/order у Postman можна додати заголовок Authorization: Bearer {{token}}, передати у тілі запиту параметри активу (символ монети, обсяг, тригерні рівні Stop Loss та Take Profit) й миттєво побачити результат обробки транзакції торговим ядром. Перевагою також є вбудовані можливості для написання автоматизованих скриптів тестування на базі JavaScript, які дозволяють здійснювати перевірку коректності відповіді на кожен запит. Наприклад, можна програмно переконатися, що при успішному виконанні торгової угоди сервер повертає статус-код 201 Created, а тіло відповіді містить

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				
Змн.	Арк.	№ докум.	Підпис	Дата		65

оновлені значення балансу віртуального гаманця. Таким чином, Postman виступає не лише як інструмент для ручного тестування, але й як повноцінна платформа для документування та адміністрування API.

Для забезпечення зручного та ефективного розповсюдження документації розробленого API використовується функціонал експорту та імпорту колекцій. Однією з ключових переваг інтеграції Postman із фреймворком NestJS є можливість автоматичного генерування документації за стандартом OpenAPI (Swagger) та її подальшого імпорту у вигляді JSON-файлу. Такий файл містить повний набір запитів, опис параметрів, маршрутів, моделей даних та змінних. Щоб імпортувати документацію в Postman, необхідно у верхньому лівому куті інтерфейсу натиснути кнопку імпорту, перейти на вкладку завантаження файлів і вибрати відповідний документ. Після аналізу файлу система автоматично створить нову колекцію, у якій міститимуться всі HTTP-запити, логічно згруповані за доменами: /market, /trade, /users та /profile. Кожен запит можна миттєво редагувати, запускати та зберігати результати для подальшого аналізу або демонстрації.

Оскільки в колекції активно використовуються динамічні змінні, такі як `{{baseUrl}}` для перемикання між локальним сервером і хостингом, `{{token}}` для збереження сесії авторизації або `{{symbol}}` для швидкої зміни криптоактиву при перевірці історичних даних ринку, у Postman було створено відповідне віртуальне середовище (Environment). Це дозволяє уникати ручного редагування кожного окремого запиту і забезпечує високу швидкість наскрізного тестування всього функціоналу. Перед вами зображення документації для роутів API, яке наочно демонструє два типи критично важливих запитів: POST для автентифікації користувача та GET для отримання поточного стану інвестиційного портфеля. У кожному запиті зафіксовано необхідні параметри та очікувані структури відповідей. Документація дає зрозуміле уявлення про те, як саме клієнтський додаток Next.js повинен формувати мережеві запити для коректної взаємодії із

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				66
Змн.	Арк.	№ докум.	Підпис	Дата		

серверною частиною платформи. Зображення виглядає наступним чином (рис. 3.7).

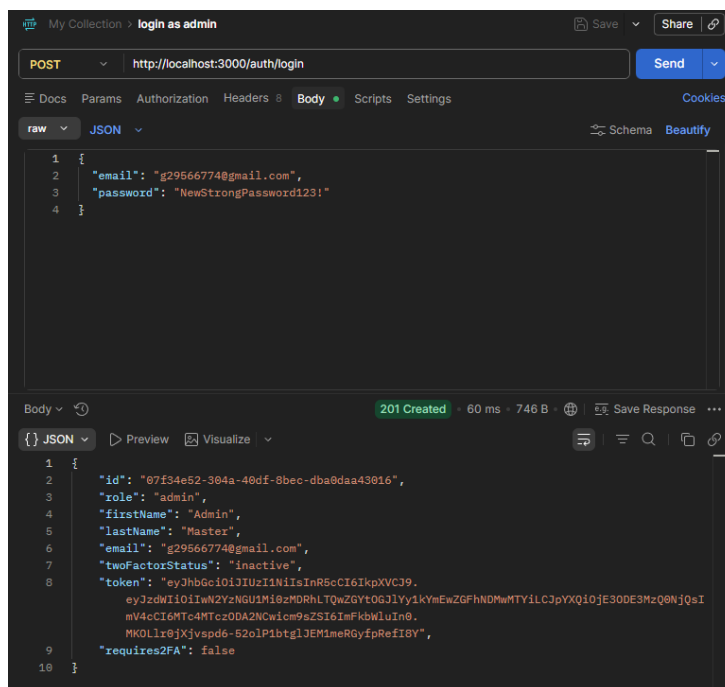


Рисунок 3.7 - Документація роутів Postman для управління профілем та авторизацією

Висновки до третього розділу

У третьому розділі було детально розглянуто архітектуру користувацького інтерфейсу та алгоритми взаємодії трейдера з веб-платформою симуляції криптовалютних торгів «Trading Simulator Platform». Додаток побудовано за сучасною моделлю клієнт-сервер, де фронтенд і бекенд мають чітко розмежовані зони відповідальності та взаємодіють як через класичний REST API, так і через високошвидкісні двосторонні з'єднання WebSockets. Особливу увагу було приділено організації маршрутизації, яка спроектована за логічним ресурсним принципом і підтримує стандартні HTTP-методи для виконання необхідних операцій. Така структура забезпечує зрозумілу, надійну та масштабовану логіку обробки запитів, що є критично важливим для стабільної роботи фінансового ядра системи та миттєвого відображення ринкових даних.

На клієнтській частині застосовано фреймворк Next.js на базі бібліотеки React, що дозволило реалізувати оптимізовану та реактивну компонентну

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				67
Змн.	Арк.	№ докум.	Підпис	Дата		

архітектуру з використанням парадигми App Router. Інтерфейс логічно поділено на спеціалізовані сторінки: головний термінал з інтерактивними графіками японських свічок (за допомогою lightweight-charts), панель управління портфелем з візуалізацією розподілу активів (через recharts) та захищені форми авторизації. Уся навігація побудована на основі маршрутизації файлової системи, яка тісно синхронізована зі структурою контролерів бекенду на NestJS. Це дозволяє зберегти єдиний архітектурний підхід до управління даними, полегшує подальшу підтримку коду та гарантує плавний перехід між торговими парами без повного перезавантаження сторінок браузера, зберігаючи поточний контекст додатка.

Окремий акцент у розділі було зроблено на механізмах автентифікації та безпеки, які базуються на використанні протоколу JWT (JSON Web Token) у поєднанні із системою підтвердження реєстрації через електронну пошту. Такий підхід забезпечує надійну безстейтову (stateless) перевірку прав доступу, повністю усуваючи необхідність зберігання активних сесій в оперативній пам'яті сервера. Серверне ядро перевіряє криптографічний підпис токена і надає дозвіл на здійснення торгових операцій або перегляд статистики лише у разі його валідності. Важливо зазначити, що механізм JWT успішно інтегровано не лише в HTTP-запити, але й у WebSocket-з'єднання, що гарантує захищену та персоналізовану трансляцію котирувань і сповіщень про спрацювання ордерів Stop Loss або Take Profit у реальному часі.

Загалом, розроблений користувацький інтерфейс у комплексі з продуманою маршрутизацією та строгими механізмами захисту формує стабільну, швидку та безпечну екосистему для симуляції криптоторгів. Важливою особливістю фронтенду є використання сучасних підходів до стилізації за допомогою Tailwind CSS, що дозволило створити преміальний візуальний досвід із застосуванням темної теми та ефектів glassmorphism. Інтерфейс спроектовано з глибоким розумінням UX-принципів: реалізовано миттєвий зворотний зв'язок при встановленні ордерів, динамічний розрахунок нереалізованих прибутків і збитків (PnL) щосекунди, а також наочне відображення підказок предиктивної AI-

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				68
Змн.	Арк.	№ докум.	Підпис	Дата		

аналітики безпосередньо на графіку. Завдяки цьому платформа є не лише технологічно досконалою, але й максимально зручною, безпечною та інтуїтивно зрозумілою для трейдерів будь-якого рівня підготовки.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				69
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У межах кваліфікаційної роботи бакалавра було спроектовано та реалізовано повнофункціональний вебзастосунок для прогнозування ринкових трендів на основі ансамблів нейромереж, інфраструктурним ядром якого стала інтерактивна платформа для симуляції торгів криптоактивами «Trading Simulator Platform». Систему побудовано на основі передового стеку технологій: модульного фреймворку NestJS для серверної частини, що дозволяє легко інтегрувати та масштабувати аналітичні алгоритми, та бібліотеки React у поєднанні з Next.js для клієнтського інтерфейсу. Завдяки використанню REST API та глибокій інтеграції протоколу WebSockets забезпечено чітке розмежування між архітектурними шарами, безперервний збір котирувань у реальному часі для живлення предиктивних моделей, а також високу загальну продуктивність і зручність підтримки системи.

З боку серверної частини, за допомогою СУБД PostgreSQL та інструментарію Prisma ORM реалізовано надійну взаємодію з реляційною базою даних. Це надало змогу не лише безпечно зберігати фінансову історію користувачів із дотриманням транзакційної цілісності для уникнення стану гонитви (race conditions), але й формувати структуровані масиви історичних даних (датасети), критично важливі для навчання та валідації ансамблів нейромереж. Використання спеціалізованої бібліотеки decimal.js забезпечило абсолютну математичну точність при розрахунку балансів під час апробації згенерованих торгових стратегій. Завдяки розділенню бізнес-логіки на контролери, сервіси та автономні фонові процеси (зокрема, для обробки тригерів Stop Loss і Take Profit), програмний код залишається гнучким і готовим до впровадження складніших інтелектуальних модулів.

Користувацький інтерфейс, побудований на базі React та архітектури App Router, гарантує швидку, реактивну взаємодію з аналітичним терміналом, зручну навігацію без перезавантаження сторінок та адаптивність візуалізації результатів прогнозування. Візуальне оформлення за допомогою Tailwind CSS із

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				70
Змн.	Арк.	№ докум.	Підпис	Дата		

застосуванням ефектів glassmorphism створює преміальний користувацький досвід. Під час розробки були успішно реалізовані всі ключові функціональні можливості: безпечна реєстрація з підтвердженням електронної пошти, JWT-автентифікація, відображення графіків японських свічок (lightweight-charts) із можливістю відстеження згенерованих штучним інтелектом трендів, управління ордерами, динамічний розрахунок PnL, візуалізація портфеля (recharts) та інтеграція предиктивної AI-аналітики.

Використання середовища Postman для документування та тестування API надало змогу проводити ефективну перевірку маршрутизованих запитів, механізмів авторизації за допомогою токенів та безпечної взаємодії під час передачі великих обсягів даних між фронтендом і бекендом. Це значно спростило процес розробки, налагодження логіки торгового ядра та інтеграції алгоритмів машинного навчання. У підсумку, результати виконання дипломної роботи підтверджують доцільність обраного технологічного підходу та архітектурних рішень. Розроблений програмний продукт є цілісною, безпечною та автономною екосистемою, яку можна розглядати як ефективне навчальне середовище для трейдерів, а також як міцний інфраструктурний полігон для подальшого впровадження ансамблів нейромереж у реальних умовах фінансового ринку.

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				71
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vranken H. Sustainability of bitcoin and blockchains. *Current Opinion in Environmental Sustainability*. 2017. Vol. 28. P. 1–9. DOI: 10.1016/j.cosust.2017.04.011 (дата звернення: 01.05.2026).
2. Obe R. O., Hsu L. S. PostgreSQL: Up and Running. Sebastopol: O'Reilly Media, 2017. 312 p. (дата звернення: 02.05.2026).
3. Prisma ORM Official Documentation. URL: <https://www.prisma.io/docs/> (дата звернення: 03.05.2026).
4. NestJS Official Documentation. URL: <https://docs.nestjs.com/> (дата звернення: 04.05.2026).
5. Decimal.js API Reference: An arbitrary-precision Decimal type for JavaScript. URL: <https://mikemcl.github.io/decimal.js/> (дата звернення: 05.05.2026).
6. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. Sebastopol: O'Reilly Media, 2020. 310 p. (дата звернення: 06.05.2026).
7. Next.js App Router Documentation. URL: <https://nextjs.org/docs> (дата звернення: 07.05.2026).
8. Vanderkam D. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript. Sebastopol: O'Reilly Media, 2019. 266 p. (дата звернення: 08.05.2026).
9. Tailwind CSS Documentation. URL: <https://tailwindcss.com/docs> (дата звернення: 09.05.2026).
10. Lightweight Charts Documentation. URL: <https://tradingview.github.io/lightweight-charts/> (дата звернення: 10.05.2026).
11. Recharts: A composable charting library built on React components. URL: <https://recharts.org/> (дата звернення: 11.05.2026).
12. JSON Web Token (JWT) Introduction. URL: <https://jwt.io/introduction/> (дата звернення: 12.05.2026).
13. Goldberg J. Learning TypeScript. Sebastopol: O'Reilly Media, 2022. 317 p. (дата звернення: 13.05.2026).

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ІЗ	Арк.
		Т. А. ВАКАЛЮК				72
Змн.	Арк.	№ докум.	Підпис	Дата		

- 14.Socket.IO Documentation: Bidirectional and low-latency communication. URL: <https://socket.io/docs/v4/> (дата звернення: 14.05.2026).
- 15.Wang V., Salim F., Moskovits P. The Definitive Guide to HTML5 WebSocket. Berkeley: Apress, 2013. 248 p. (дата звернення: 15.05.2026).
- 16.Freeman A. Pro React 16. Berkeley: Apress, 2019. 768 p. (дата звернення: 16.05.2026).
- 17.Casciaro M., Mammino L. Node.js Design Patterns. Birmingham: Packt Publishing, 2020. 664 p. (дата звернення: 17.05.2026).
- 18.Fielding R. T., Taylor R. N. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*. 2002. Vol. 2, No 2. P. 115–150. DOI: 10.1145/514183.514185 (дата звернення: 18.05.2026).
- 19.TensorFlow.js: Machine Learning for the Web and Beyond / D. Smilkov et al. *Proceedings of Machine Learning and Systems*. 2019. Vol. 1. P. 309–321. (дата звернення: 19.05.2026).
- 20.Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2003. 560 p. (дата звернення: 20.05.2026).
- 21.Hickson I. The WebSocket API. W3C Recommendation. 2012. URL: <https://www.w3.org/TR/websockets/> (дата звернення: 21.05.2026).
- 22.Suzuki H. The Internals of PostgreSQL. URL: <https://www.interdb.jp/pg/> (дата звернення: 22.05.2026).
- 23.Fowler M. CQRS Pattern. 2011. URL: <https://martinfowler.com/bliki/CQRS.html> (дата звернення: 23.05.2026).
- 24.Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol: O'Reilly Media, 2021. 614 p. (дата звернення: 24.05.2026).
- 25.Poulton N. Docker Deep Dive. [S. l.]: Leanpub, 2021. 248 p. (дата звернення: 25.05.2026).

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ІПЗ	Арк.
		Т. А. ВАКАЛЮК				73
Змн.	Арк.	№ докум.	Підпис	Дата		

- 26.Rad B. B., Bhatti H. J., Ahmadi M. An Introduction to Docker and Analysis of its Performance. *International Journal of Computer Science and Network Security*. 2017. Vol. 17, No 3. P. 228–235. (дата звернення: 26.05.2026).
- 27.NGINX Official Documentation. URL: <https://nginx.org/en/docs/> (дата звернення: 27.05.2026).
- 28.Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p. (дата звернення: 28.05.2026).
- 29.Riva M. Real-World Next.js: Build scalable, high-performance, and modern web applications. Birmingham: Packt Publishing, 2022. 366 p. (дата звернення: 29.05.2026).
- 30.Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. 2015. P. 1–30. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 30.05.2026).
- 31.Postman Learning Center: Documentation and API Testing. URL: <https://learning.postman.com/docs/getting-started/introduction/> (дата звернення: 31.05.2026).

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				74
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		О. О. ГАНЧЕВСЬКИЙ			ІПЗ – 121 – 25 - ПЗ	Арк.
		Т. А. ВАКАЛЮК				75
Змн.	Арк.	№ докум.	Підпис	Дата		

Додаток А

Лістинг users.service.ts:

```
import { randomUUID } from 'crypto';
import { BanUserDTO, CreateUserDTO, ViewUserDTO } from './dto';
import {
  BadRequestException,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import { PrismaService } from '../prisma';
import { mapUserRoleToDb, UserViewMapper } from './mappers';
import { UserStatus } from '../generated/prisma/enums';
import { PasswordResetService } from './password-reset.service';
import { HelloEmailService } from './email';

@Injectable()
export class UsersService {
  private readonly mapper = new UserViewMapper();
  constructor(
    private readonly helloEmailService: HelloEmailService,
    private readonly passwordResetService: PasswordResetService,
    private readonly prisma: PrismaService,
  ) {}

  async create(data: CreateUserDTO): Promise<ViewUserDTO> {
    await this.checkEmail(data.email);
    const newUserId = randomUUID();

    const user = await this.prisma.user.create({
      data: {
        id: newUserId,
        role: mapUserRoleToDb(data.role),
        firstName: data.firstName,
        lastName: data.lastName,
        email: data.email,
        status: UserStatus.active,
        createdBy: newUserId,
      },
    });

    const reset = await this.passwordResetService.createOrReplace(user.id);
    await this.helloEmailService.send({
      ...reset,
      email: user.email,
    });
  }
}
```

		О. О. ГАНЧЕВСЬКИЙ			ІІЗ – 121 – 25 - ІІЗ	Арк.
		Т. А. БАКАЛЮК				76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    name: user.firstName,
  });
  return this.mapper.mapOne(user);
}

async get(): Promise<ViewUserDTO[]> {
  const data = await this.prisma.user.findMany({});
  return this.mapper.mapMany(data);
}

async getOne(id: string): Promise<ViewUserDTO> {
  const data = await this.prisma.user.findFirst({
    where: { id },
  });
  if (!data) {
    throw new NotFoundException('User not found');
  }
  return this.mapper.mapOne(data);
}

async update(id: string, data: CreateUserDTO): Promise<ViewUserDTO> {
  const user = await this.prisma.user.findFirst({ where: { id } });
  if (!user) {
    throw new NotFoundException('User not found');
  }

  if (user.email !== data.email) {
    await this.checkEmail(data.email);
  }

  const updatedUser = await this.prisma.user.update({
    where: { id },
    data: {
      email: data.email,
      firstName: data.firstName,
      lastName: data.lastName,
      role: mapUserRoleToDb(data.role),
    },
  });

  return this.mapper.mapOne(updatedUser);
}

async ban(id: string, data: BanUserDTO): Promise<ViewUserDTO> {

```

		О. О. ГАНЧЕВСЬКИЙ			ІІЗ – 121 – 25 - ІІЗ	Арк.
		Т. А. БАКАЛЮК				77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const user = await this.prisma.user.findFirst({ where: { id } });
if (!user) {
  throw new NotFoundException('User not found');
}
const updatedUser = await this.prisma.user.update({
  where: { id },
  data: {
    status: data.banned ? UserStatus.banned : UserStatus.active,
  },
});

return this.mapper.mapOne(updatedUser);
}

private async checkEmail(email: string): Promise<void> {
  const user = await this.prisma.user.findFirst({
    where: { email: { equals: email, mode: 'insensitive' } },
    select: { id: true },
  });
  if (user) {
    throw new BadRequestException('User with the same email already exists');
  }
}

```

Лістинг market.service.ts:

```

import { Injectable, OnModuleInit, OnModuleDestroy } from '@nestjs/common';
import WebSocket from 'ws';
import { MarketGateway } from './market.gateway';
import { BinanceKlinePayload } from './dto/binance-ws.interface';
import { MarketStreamMessageDTO } from './dto/market-stream.dto';
import { HttpService } from '@nestjs/axios';
import { firstValueFrom } from 'rxjs';
import { SymbolsService } from './symbols.service';
import { PredictService } from './ai';
import { PriceService } from './price.service';
import { BinanceTicker24h } from './dto';

```

```

interface CustomWebSocket extends WebSocket {
  connectionKey?: string;
}

```

@Injectable()

```

export class MarketService implements OnModuleInit, OnModuleDestroy {
  private binanceSockets: WebSocket[] = [];
}

```

		О. О. ГАНЧЕВСЬКИЙ			ІІЗ – 121 – 25 - ІІЗ	Арк.
		Т. А. БАКАЛЮК				78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

private activeConnections = new Set<string>();

private readonly SUPPORTED_INTERVALS = ['1m', '15m', '1h', '1d'];

constructor(
  private readonly marketGateway: MarketGateway,
  private readonly httpService: HttpService,
  private readonly symbolsService: SymbolsService,
  private readonly predictService: PredictService,
  private readonly priceService: PriceService,
) {}

async onModuleInit() {
  const symbols = await this.symbolsService.getActiveSymbols();

  symbols.forEach((symbol) => {
    this.SUPPORTED_INTERVALS.forEach((interval) => {
      this.connectToBinanceStream(symbol, interval);
    });
  });
}

onModuleDestroy() {
  this.binanceSockets.forEach((ws) => ws.close());
}

public async getSupportedSymbols(): Promise<string[]> {
  const symbols = await this.symbolsService.getActiveSymbols();
  return symbols.map((s) => s.toUpperCase());
}

public getSupportedIntervals(): string[] {
  return [...this.SUPPORTED_INTERVALS];
}

public async addTrackedSymbol(symbol: string, name?: string) {
  const newSymbol = await this.symbolsService.addSymbol(symbol, name);

  this.SUPPORTED_INTERVALS.forEach((interval) => {
    this.connectToBinanceStream(newSymbol.symbol, interval);
  });

  return newSymbol;
}

```

		О. О. ГАНЧЕВСЬКИЙ			ІІЗ – 121 – 25 - ІІЗ	Арк.
		Т. А. БАКАЛЮК				79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

}

public async getHistory(
    symbol: string,
    interval: string = '1d',
): Promise<
{
    time: number;
    open: number;
    high: number;
    low: number;
    close: number;
    volume: number;
>[]
> {
    const url =
`https://api.binance.com/api/v3/klines?symbol=${symbol.toUpperCase()}&interval=${
interval}&limit=1000`;
    const response = await firstValueFrom(
        this.httpService.get<(string | number)[>](url),
    );

    return response.data.map((item) => ({
        time: Math.floor(Number(item[0]) / 1000),
        open: parseFloat(String(item[1])),
        high: parseFloat(String(item[2])),
        low: parseFloat(String(item[3])),
        close: parseFloat(String(item[4])),
        volume: parseFloat(String(item[5])),
    }));
}

private connectToBinanceStream(symbol: string, interval: string) {
    const connectionKey = `${symbol}_${interval}`;
    if (this.activeConnections.has(connectionKey)) return;

    const wsUrl = `wss://stream.binance.com:9443/ws/${symbol}@kline_${interval}`;
    const ws = new WebSocket(wsUrl) as CustomWebSocket;
    ws.connectionKey = connectionKey;

    this.binanceSockets.push(ws);
    this.activeConnections.add(connectionKey);

    ws.on('open', () => {

```

		О. О. ГАНЧЕВСЬКИЙ			ІІІ – 121 – 25 - ІІІ	Арк.
		Т. А. БАКАЛЮК				80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

console.log(
  `[Binance WS] Підключено: ${symbol.toUpperCase()} (${interval})`,
);
});

ws.on('message', (data: WebSocket.RawData) => {
  const payload = JSON.parse(
    (data as Buffer).toString(),
  ) as BinanceKlinePayload;

  this.priceService.updatePrice(payload.s, parseFloat(payload.k.c));

  const formattedMessage: MarketStreamMessageDTO = {
    symbol: payload.s,
    interval: payload.k.i,
    kline: {
      time: payload.k.t,
      open: parseFloat(payload.k.o),
      high: parseFloat(payload.k.h),
      low: parseFloat(payload.k.l),
      close: parseFloat(payload.k.c),
      volume: parseFloat(payload.k.v),
      isClosed: payload.k.x,
    },
  };

  this.marketGateway.broadcastMarketData(formattedMessage);
});

ws.on('error', (error) => {
  console.error(
    `[Binance WS] Помилка потоку ${symbol} (${interval}):`,
    error,
  );
});

ws.on('close', () => {
  this.binanceSockets = this.binanceSockets.filter((s) => s !== ws);
  this.activeConnections.delete(connectionKey);

  this.symbolsService
    .checkIsActive(symbol)
    .then((isActive) => {
      if (isActive) {

```

		О. О. ГАНЧЕВСЬКИЙ			ІІЗ – 121 – 25 - ІІЗ	Арк.
		Т. А. ВАКАЛЮК				81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        console.log(
            `[Binance WS] Перепідключення ${symbol} (${interval})...`,
        );
        setTimeout(
            () => this.connectToBinanceStream(symbol, interval),
            5000,
        );
    }
})
.catch((err: unknown) =>
    console.error(
        `[Binance WS] Помилка під час перевірки статусу ${symbol}:`,
        err,
    ),
);
});
}

```

```

public async removeTrackedSymbol(symbol: string) {
    const normalizedSymbol = symbol.toLowerCase().trim();

    await this.symbolsService.deactivateSymbol(normalizedSymbol);

    const socketsToClose = this.binanceSockets.filter((ws) =>
        (ws as CustomWebSocket).connectionKey?.startsWith(`${normalizedSymbol}_`),
    );

    socketsToClose.forEach((ws) => ws.close());
}

public async getInstantPrediction(symbol: string, interval: string = '1d') {
    const history = await this.getHistory(symbol, interval);

    if (history.length < 60) {
        return { trend: 'neutral', confidence: 0 };
    }

    const priceBuffer = history.slice(-60).map((h) => h.close);
    const currentPrice = priceBuffer[priceBuffer.length - 1];

    const prediction = await this.predictService.predictLstm(
        priceBuffer,
        currentPrice,
        interval,
    );
}

```

		О. О. ГАНЧЕВСЬКИЙ			ІІІ – 121 – 25 - ІІІ	Арк.
		Т. А. ВАКАЛЮК				82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

);

return prediction;
}

public async getBulk24hStats(symbols: string[]) {
  if (!symbols || symbols.length === 0) return [];

  const formattedSymbols = JSON.stringify(
    symbols.map((s) => s.toUpperCase()),
  );
  const url =
`https://api.binance.com/api/v3/ticker/24hr?symbols=${formattedSymbols}`;

  try {
    const response = await firstValueFrom(
      this.httpService.get<BinanceTicker24h[]>(url),
    );

    return response.data.map((ticker) => ({
      symbol: ticker.symbol,
      currentPrice: parseFloat(ticker.lastPrice),
      high24h: parseFloat(ticker.highPrice),
      low24h: parseFloat(ticker.lowPrice),
      priceChangePercent: parseFloat(ticker.priceChangePercent),
    }));
  } catch (error) {
    console.error(
      '[Binance API] Помилка завантаження 24h статистики:',
      error,
    );
    return [];
  }
}

```

		О. О. ГАНЧЕВСЬКИЙ			ІІЗ – 121 – 25 - ІІЗ	Арк.
		Т. А. БАКАЛЮК				83
Змн.	Арк.	№ докум.	Підпис	Дата		